REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED *(From - To)*
30 July 2017	Tutorial/Manual	01 June 2017 - 30 July 2017

4. TITLE AND SUBTITLE
Tutorial for Thermophysics Universal Research Framework

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
S. Araki, D. Bilyeu, J. Koo, R. Martin, J. Tran, K. Kawashima

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
Q1NC

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory (AFMC)
AFRL/RQRS
1 Ara Drive
Edwards AFB, CA 93524-7013

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory (AFMC)
AFRL/RQR
5 Pollux Drive
Edwards AFB, CA 93524-7048

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-RQ-ED-OT-2017-186

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. PA Clearance Number: 17441 Clearance Date: 10 July 2017

13. SUPPLEMENTARY NOTES
Prepared in collaboration with ERC. The U.S. Government is joint author of the work and has the right to use, modify, reproduce, release, perform, display, or disclose the work.

14. ABSTRACT
TURF development has been ongoing at AFRL/RQRS since late 2011 and has reached a level of development and testing where the core data-structures, interface, and select operators are ready to be used by external collaborators familiar with plasma physics and c++. To facilitate bringing these collaborators up to speed in writing modules for TURF, the TURF Infrastructure Release (TURF-IR) has the capabilities in place to demonstrate kinetic and fluid simulations, including several example input files to provide convenient starting points for the development of additional physics capabilities. Though the framework has been designed in part to facilitate the creation of modules that replicate the functionality of Coliseum/HPHall, it must be emphasized that the TURF-IR is intended to stimulate academic collaboration and does not provide equivalent real-world capabilities to the Coliseum/HPHall suite [1]–[3]. As such, this software by itself cannot be used to design or analyze real systems.

15. SUBJECT TERMS
N/A

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			D. Bilyeu
Unclassified	Unclassified	Unclassified	SAR	149	19b. TELEPHONE NUMBER *(Include area code)* N/A

TUTORIAL FOR THERMOPHYSICS UNIVERSAL RESEARCH FRAMEWORK

– TURF-IR 2017a –

May 2017

Samuel J. Araki[1], Robert Martin[2], David Bilyeu[2],
Justin Koo[2], Jonathan Tran[1], Kari A. Kawashima[1]

[1]ERC Inc.
[2]AFRL/RQRS

CONTENTS

CHAPTER 1

OVERVIEW

Robert Martin and Justin Koo

Contents

1.1 Introduction

TURF development has been ongoing at AFRL/RQRS since late 2011 and has reached a level of development and testing where the core data-structures, interface, and select operators are ready to be used by external collaborators familiar with plasma physics and c++. To facilitate bringing these collaborators up to speed in writing modules for TURF, the TURF Infrastructure Release (TURF-IR) has the capabilities in place to demonstrate kinetic and fluid simulations, including several example input files to provide convenient starting points for the development of additional physics capabilities. Though the framework has been designed in part to facilitate the creation of modules that replicate the functionality of Coliseum/HPHall, it must be emphasized that the TURF-IR is intended to stimulate academic collaboration and does not provide equivalent real-world capabilities to the Coliseum/HPHall suite [1]–[3]. As such, this software by itself cannot be used to design or analyze real systems.

1.2 Core Structure

TURF is designed around a basic tree hierarchical object structure. TURF objects are built around this core "General Service Object" or "GSObject" that facilitates construction of object trees and allows branches of the tree to be recursively copied across disparate memory spaces such as from the CPU to GPU or across the message passing interface (MPI) without losing their structural integrity. Because TURF objects are all derived from this basic type, in addition to the core functionality required for this recursive data movement, the GSObject also facilitates recursive auto-documentation of runtime object structure.

The objects of TURF can be divided between data objects and operation objects. The number of data objects are intentionally restricted to provide a basic common skeleton of data storage on which a broad set of operations can be applied. These basic data objects with minimal independent functionality facilitate code reuse by providing a common basis though which operations interact. They also help simplify parallel communication by minimizing the set of disparate objects that must be transferred. The operation objects define the key application programming interface through which developers are encouraged to interact with the framework. They are intended to represent compact mathematical operations and consist of 3 key components. Every operation includes an "init" function that is passed a map of parameter-key values. It is the operation's responsibility to parse this map to initialize all

the local information, create data-structures as needed, and query the existing object hierarchy for references to additional required objects needed to function. For some operations, such as initial conditions, this initialization is all that is required. For operations that evolve the problem solution, an "apply" function is defined. This function is called as the framework iterates through a sequence of operations to perform a specific function on the data. Whenever possible, this functionality should be further broken down into one or more "core" functions that represent the functionality in a data independent parallel form that is designed to be agnostic to whether the core is applied sequentially on a single process, in parallel through threads or as implemented by OpenMP, or in parallel on an accelerator such as the GPU. The use of OpenMP and GPU are currently considered experimental and are not widely implemented or tested.

Though additional data-structures may be created and inherited from those already included in the TURF-IR, this practice is discouraged in favor of using existing data-structures to the greatest extent possible so that operations apply in the largest context possible. Both the organizational layout of the data as well as the sequence of operations performed on the data are then constructed at runtime facilitating rapid modular algorithm design. As a result of this highly modular design, library modules of operations can be included or omitted without impacting the functionality of other modules. This "plug-in" model implies that, if a particular release of TURF is missing a module (e.g. a C-R physics module), there are no hooks to indicate that a particular module is missing. In this way, reverse engineering the functionality of a module from the generic interfaces defined by the framework is impossible. For an authorized developer, however, it is critical that the interfaces between their modules and the framework are well-defined. Actually having a copy of the TURF-IR permits them to test the compatibility of their modules within the framework and helps ensure data-structure compatibility and adherence to the module interface for delivered code.

1.3 Key Data-structures

Creating modules that interface efficiently with TURF requires a detailed understanding of the layout of data in memory (i.e. the data-structures). These underlying data-structures are largely tied to the sort of data being stored (particle/field data) and the sort of mesh on which it is being stored (mesh-free/structured mesh/unstructured mesh) but are all based on a custom multi-dimensional matrix object called gMatrix, a GSObject encapsulation and extension of the matrix objects used extensively in AFRL/RQRS's prior research codes [4]–[8]. As a fundamental data object, it is generic templated container class. The mesh classes of SMesh (structured) and UMesh (unstructured) are compound objects of header information and gMatrix objects based on generic mesh classes used also extensively by AFRL/RQRS over the last five years in numerous PhD and Masters theses. In addition to mesh objects, the TURF-IR includes examples of basic particle and field data objects with similar header/gMatrix compound structure.

1.4 Basic Functionality

For this version of the TURF-IR, the computational domain is centered around a constant spaced global Cartesian coordinate system. Though this "LogicalWorld" object is derived from a more general "World" object, the current version of the framework assumes the existence of the global real to logical coordinate system to facilitate domain patching automation while varying resolution. Active regions of the global coordinate system are defined by "LogicalDomain" objects. These are essentially intended to be discrete axis aligned blocks for the sake of domain decomposition. Computation on every domain should be able to run independent of the others except for discrete synchronization points at which time patches between domains are guaranteed to have completed. These synchronization points are the end of computational stages.

Once the world and domains have been created, various "Operations" are applied. These operations can create additional data objects to attach to the domain, set initial and boundary conditions, solve sets of equations, write output, or any number of other manipulations of the data within the framework so long as they may be applied for one domain at a time independently. Communication between domains is restricted to stage boundaries to preserve this independence. The intent of this is two pronged. The first goal is to encourage as much fine grain parallelization as possible. Extremely broad definition of an operation is also intended to avoiding locking the framework to a

specific set of applications due to more rigidly defined interfaces and phases of computation. The function of the framework is then an extremely generic statement of, "There exists a set of data on which a sequence of parallel operations may be applied that can be broken down into discrete stages between which communication between the datasets may be performed." In general, the world typically advances in computational time looping through the operations, but this functionality is only included to assist in timestep synchronization between domains as this is a commonly required functionality not easily obtainable in a domain independent manner. The form of the time advancement, whether implicit, explicit, or iteration towards steady state, must be defined as built into the sequence of operations rather than a pre-selected as a trait of the framework.

CHAPTER 2

TURF INFRASTRUCTURE RELEASE 2016

Robert Martin, Samuel J. Araki, Jonathan Tran, and David Bilyeu

Contents

2.1 Capabilities

Robert Martin and Samuel J. Araki

TURF-IR v1.0 [9] included operators necessary to run four fundamental problems. These include a free-molecular flow in a specular box (heatbath), a plasma in a grounded box, one-dimensional normal shock wave, and one-dimensional collisionless electrostatic shock as summarized in Table 2.1. This level of functionality was originally released to demonstrate the capabilities of the framework and provide a starting point for enabling collaborators to develop new models and algorithms for the framework. For the 2016 release, the original tutorials have been updated to remain consistent with the evolution of the overall framework while attempting to maintain backwards compatibility to the greatest extent possible. Some of the operators included in 2016 have been superseded by newer versions with enhanced capabilities as described in the TURF 2017a capabilities document. They are included to maintain compatibility with the previous examples, but attempts should be made to migrate to the newer 2017a capability if possible for enhanced functionality and performance in the future. The list of operators included in TURF-IR v2016 is provided in Tables 2.2 and 2.3.

Table 2.1: Tutorials provided for TURF-IR v2016.

Folder	Description of Problem	Type of Solver	Section
Heatbath	Free molecular flow in a specular box	Particle	2.2 and 2.3
GroundedBox	Plasma in a grounded box	PIC	2.4
1DShock	One-dimensional normal shock wave	DSMC	2.5
CollisionlessShock	One-dimensional electrostatic shock	Vlasov	2.6 and 2.7

Table 2.2: Summary of operations included in TURF-IR v2016.

Module	Operation	Description
DSMC	`SPDistDSMCConstantBCOp`	Injection of constant weight particles
DSMC	`SPDistDSMCConstantICOp`	Initial distribution of particles inside the domain
DSMC	`SPDistDSMCOp`	DSMC collision calculation
DSMC	`SPDistDSMCSample2Op`	Blend running average of field data
DSMC	`SPDistDSMCSampleOp`	Sample weights and number of particles
Field	`LogicalBCConstantOp`	Set value of cell centers in box every iteration
Field	`LogicalBCXtrapOp`	Set a physical boundary to extrapolation
Field	`LogicalFieldAddOp`	Add one field variable to another
Field	`LogicalFieldScalarMulOp`	Multiply field by scalar constant
Field	`LogicalFieldSetOp`	Set field values to constant
Field	`LogicalFieldVolumetricMulOp`	Multiply or divide by cell volume
Field	`LogicalGradientCellCenterOp`	Calculate the gradients of a field vector
Field	`LogicalNodeGradientOp`	Calculate node-centered gradient of cell center field
Field	`LogicalNormOp`	Calculate L^p-norm of field variable
Field	`LogicalPoissonBoltzmannStrip1DOp`	Solve for the electric field assuming a Boltzmann electrons
Field	`LogicalPoissonStripOp`	Red/Black line realxing Poisson solve
Field	`LogicalResidualOp`	Calculate residual of Poisson solve
Particle	`SPDistBCSpecOp`	Specularly reflecting boundary condition
Particle	`SPDistCellIDOp`	Find cell ID associated with particle location
Particle	`SPDistCombineOp`	Unify the particles from different distributions
Particle	`SPDistConstantBCOp`	Add Particles in box with uniform cell density via weights
Particle	`SPDistConstantICOp`	Create particle distribution & add particles via SPDistConstantBCOp
Particle	`SPDistDensityToFieldOp`	Sum real and computational particles/cell to field
Particle	`SPDistESPushOp`	Electrostatic particle push using node electric field
Particle	`SPDistMoveOp`	Linear particle push

Table 2.3: Summary of operations included in TURF-IR v2016 (Continued).

Module	Operation	Description
Particle	`SPDistPatchOp`	Inter-domain particle patch
Particle	`SPDistSortOp`	Sort particles for cellID
Particle	`SPDistSplitOp`	Split particle distribution into two by cell ID flag
Particle	`SPDistToFieldOp`	Sum particle charges to field entry
Plotting	`LogicalFieldWrite1DOp`	Write to output files for line plots
Plotting	`LogicalFieldWriteVTKOp`	Write to output files for 3D plots in .vts format
Plotting	`LogicalFieldWriteVTKROp`	Export the field data in .vtr format
Plotting	`VolumeRenderOp`	Single cubic domain realtime volume rendering
Utility	`CriteriaStageOp`	Continue to next stage if quantity below criteria
Utility	`NextStageOp`	Continue to next stage
Vlasov	`CreateVlasovVariableOp`	Create new Vlasov fluid variable
Vlasov	`LogicalBCVlasovExtrapolateOp`	Set a velocity boundary conditions to extrapolation
Vlasov	`LogicalVlasov2DWriterOp`	Export a 2D phase-space plot
Vlasov	`LogicalVlasovCalcFluidVariablesOp`	Calculate field variables given a velocity distribution
Vlasov	`LogicalVlasovFluidBoltzmannSetOp`	Set initial conditions of a Vlasov field using a Boltzmann distribution
Vlasov	`LogicalVlasovFluidConstantICOp`	Create or initialize a new Vlasov fluid
Vlasov	`Vlasov1D1VSLOp`	Advect a Vlasov variable using the Semi-Lagrangian method
Vlasov	`VlasovMetricsOp`	Export Vlasov metric data for mass and energy conservation

2.2 Heatbath 1: Initialization

Jonathan Tran

2.2.1 Introduction

This tutorial is the first of two parts which gives an overview of the heatbath example in TURF. Prior to running this example, you should already have installed TURF and verified that it was installed properly. For information on this, please read README located in src-TURF. All relevant files can be located in `tutorial-TURF/TURF-IR_2016/Heatbath/Heatbath_Part1`. You should see several files with the `.list` extension, which act as the scripting files for TURF. In addition to the TURF software you will also need a scientific visualization software that can read VTK files such as Paraview[1] or VisIt.[2] Also useful is a text file comparison utility such as Meld[3] or diff.

The heatbath example studies particles undergoing thermal expansion confined within a box. As mentioned before, this tutorial is the first of two parts. We will discuss setting up a coordinate system, logical domain and other necessary databases required for a TURF simulation in Section 2.2.2. Section 2.2.3 demonstrates the simplest TURF simulation and explains the use of World-Rank.html for visualizing the simulation space. Lastly, Section 2.2.4 details the addition of particles to the domain and the operations necessary for TURF to output data compatible with VisIt [10]. In TURF Heatbath Example Part 2, we will discuss the details of moving particles around the domain and having these particles specularly reflect off the boundary of the domain. Lastly we will construct the same simulation using multiple domains.

2.2.2 Defining Global Parameters

Running the TURF executable in the working directory will have TURF search for the default script file, `world.list`, and parses it automatically. In this example, `world.list` is a symbolic link to the file `world.heatbath.list`. We can take a look at the script by opening either file. We begin by defining the world and giving it a name. This name is arbitrary and can be anything. We then have a block of commented lines which are calls to different operation files. Each `operation.list` file when uncommented will run a different example, building on itself and adding functionality. Over the course of this tutorial we would like to elaborate on the commands used to achieve this functionality. The list of operations used in this tutorial is provided in Table 2.4.

```
DEFINE WORLD
      NAME = Heatbath-Example
############################################################
### Initially all turned off - Code does not advance ###
### ###
############################################################
### Timestep advances but code does nothing ###
### OP_FILE = operations.null.list ###
############################################################
### Add a particle distribution object with some ###
### particles in a box ###
### OP_FILE = operations.addparticles.list ###
############################################################
### Write vtk field output files periodically ###
### OP_FILE = operations.writeoutput.list ###
############################################################
```

We then define the world coordinate system, time step, field names and stages used in the example. TURF is written to assume all units are in MKS. With this in mind, the cell size is 100 μm and our time step is 1 ns. The length of the simulation is defined by the `END_TIME` of 250 ns. Dividing `END_TIME` by `START_DT` will give us

[1] www.paraview.org
[2] https://visit.llnl.gov
[3] http://meldmerge.org

Table 2.4: Summary of operations listed in `operations.writeoutput.list`.

Stage	Operation	Description
INITIALIZE	`SPDistConstantICOp`	Initial particle distribution
MOVEOP	`NextStageOp`	Continue to next stage
	`SPDistDensityToFieldOp`	Sum particles per cell for field entry
	`SPDistSortOp`	Sort particles according to cell ID
	`LogicalFieldSetOp`	Initialize the field parameters
POSTOP	`LogicalFieldWriteVTKOp`	Write to output files for 3D plots

the number of iterations in the simulation, 250. The heatbath example uses two fields named as `NHe` and `CNHe`, which store the physical and computational numbers of Helium particles per cell, respectively. This example also has two stages named `INITIALIZE` and `MOVE`. A stage is a communication synchronization point after which all of the domains within the simulation can vote on whether to proceed to the next stage or repeat the current stage. This synchronization is important because different processes may finish operating on their domains before other processes do. Failure to properly synchronize may cause the simulation to produce incorrect results. Note that the names for fields and stages are only labels similarly to the world name and do not refer to any existing information in the code. However, if used elsewhere in the code it is important to reference the same name.

```
      COORDINATES = cartesian
      ORIGIN = (0.0,0.0,0.0)
      DELTA = (100.0e-6,100.0e-6,100.0e-6)
      END_TIME = 250.0e-9
      START_DT = 1.0e-9
      FIELDS = [NHe, CNHe]
      STAGES = [INITIALIZE, MOVE]
END WORLD
```

Lastly we define our domain. The location of the domain is relative to the origin of the world coordinate system. The mesh spacing is global to the coordinate system and must be the same across all domains. In this example our domain is a cube with a length of 3.2 mm with a mesh spacing of 0.1 mm in all directions.

```
#############################################################################
## Domain Geometry
#############################################################################
DEFINE DOMAIN DOM000
      BOUND_LO = (0.0,0.0,0.0)
      BOUND_HI = (3.2e-3,3.2e-3,3.2e-3)
END DOMAIN
```

2.2.3 Simulation with No Operations

operations.null.list

The first example details the creation of a domain, followed by 250 iterations of nothing. To run this case, we uncomment the following line in the `world.list` file.

```
#########################################################
### Timestep advaces but code does nothing ###
         OP_FILE = operations.null.list ###
#########################################################
```

By doing so we call the `operations.null.list` file which defines the different operations used within the stages of the simulation. You will notice that for both the initialize stage and the move stage, there exist only one operation of the type `NextStageOp`. This operation simply tells the code to continue onto the next stage.

```
DEFINE STAGE INITIALIZE
DEFINE OPERATION
# Default Criteria to Proceed to the Next Stage
      TYPE = NextStageOp
END OPERATION
END STAGE INITIALIZE
################################################################################

DEFINE STAGE MOVE
DEFINE OPERATION
# Default Criteria to Proceed to the Next Stage
      TYPE = NextStageOp
END OPERATION
END STAGE MOVE
```

It is important to remember that the stage names initialize and move are just that, only names. Despite being named initialize, this stage is called every iteration and the GPU cores must sync before moving onto the move stage. Whether it is an initial operation or one that occurs iteratively is hard-coded in the operation source code.

World-Rank.html

When TURF is run, a html file named `World-Rank.html` is automatically generated. When opened, the user can view the object hierarchy of the example. At the base of the tree is the logical world, which was named Heatbath-Example. The branches include `GSObject` named `GSMemberVector` which have the functionality of a vector and can be used by the GPU, there is a material database, a logical domain and the coordinate system defined by the `world.list` file. It is possible to expand the hierarchy to investigate any underlying databases or arrays which are automatically generated. Another useful feature is the visualization of the simulation environment. For this current example, there exists only a single domain as shown by the blue cube. The surrounding gray region is a layer of three ghost cells which are automatically generated when the domain is formed.

By selecting on the visualization and pressing the 'm' key, we can cycle through volume view, line view and point view. This is useful for visualizing objects within the domain as we will see in the later examples. The usefulness of the `World.Rank` output will become evident as we add functionality to example. The object hierarchy and visualization will allow us to directly see the changes we have made in the course of this tutorial.

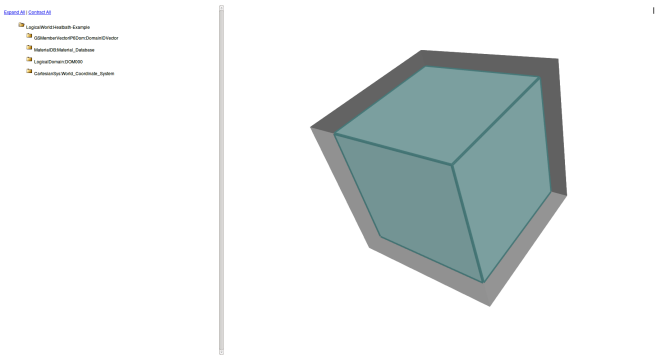


Fig. 2.1: World-Rank(0).html with contracted database hierarchy.

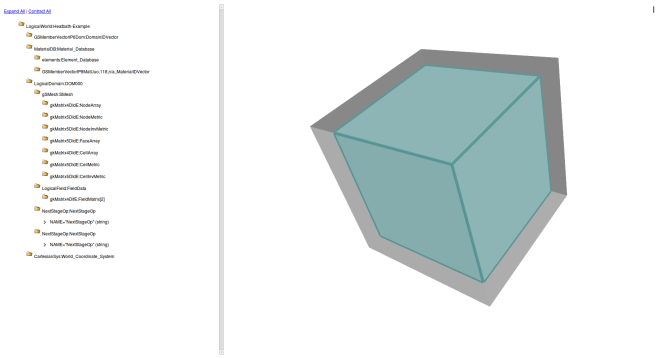


Fig. 2.2: World-Rank(0).html with expanded database hierarchy.

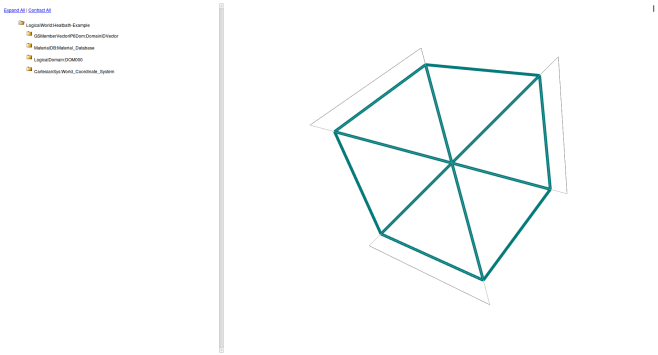


Fig. 2.3: World-Rank(0).html in line view.

2.2.4 Adding Particles

The following example creates a distribution of helium particles to fill a portion of the domain. We will then observe the changes made in the `World.Rank` file and learn how to output the data in VTK format so it can be studied using a visualization software such as VisIt.

operations.addparticles.list

To change the example we simply call a different `operation.list` file. We will do this by commenting out the previous op_file `operations.null.list` and also uncommenting the next line named `operations.addparticles.list` as shown below.

```
############################################################
### Timestep advaces but code does nothing ###
### OP_FILE = operations.null.list ###
############################################################
### Add a a particle distribution object with some ###
### particles in a box ###
         OP_FILE = operations.addparticles.list ###
############################################################
```

If we open both operations files with meld, we can directly compare changes between these two files, we can see that there is one additional operation defined of the type `SPDistConstantICOp` shown below. This operation defines a box with upper and lower boundary and distributes particles with a given number density and temperature.

The particles themselves will have a given charge, mass (in units of proton mass) and drift velocity. This specific operation creates an initial condition, so it will only do something when it is first read. Note that `FILL_RATIO` is roughly the percentage of particles to fill SPDist class object.

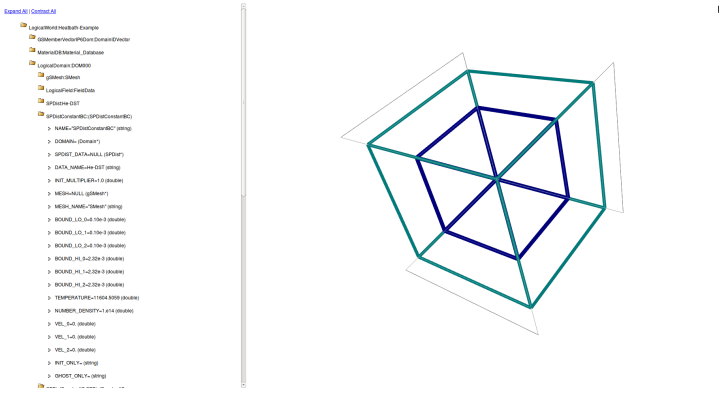


Fig. 2.4: World-Rank(0).html visualizing both the domain and region in which particles will be distributed. You may notice the code automatically generated objects when SPDistConstantICOp is called.

```
DEFINE OPERATION
       TYPE = SPDistConstantICOp
       SPDIST_DATA_NAME = He-DST
       MAX_NP = 1280000
       FILL_RATIO = 0.25
       BOUND_LO = (0.10e-3,0.10e-3,0.10e-3)
       BOUND_HI = (2.32e-3,2.32e-3,2.32e-3)
       TEMPERATURE = 11604.5059 # 1ev
       NUMBER_DENSITY = 1.e14
       Z = 0
       MASS = 4.0 Mp
       VEL = (0.,0.,0.)
END OPERATION
```

Since there are no push operations, the particles will remain at their locations for the length of the simulation. Looking at the `World-Rank.html` we can see the bounding box for which the particles will be distributed within.

operations.writeoutput.list

To run the next example, open the world.list file again. Comment the line `OP_FILE = operations.addparticles.list` and uncomment the line `OP_FILE = operations.writeoutput.list`, similarly as before. This simulation is exactly the same as the previous except now we will output the particle density distribution in a VTK format compatible with visualization software such as VisIt. This output is generated every five iterations. To do so we will use the following operations in the move stage:

```
##############################
## Sum to Fields for Output ##
##############################
DEFINE OPERATION
       TYPE = LogicalFieldSetOp
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = CNHe
       VALUE = 0.0
```

```
END OPERATION
DEFINE OPERATION
       TYPE = LogicalFieldSetOp
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = NHe
       VALUE = 0.0
END OPERATION
DEFINE OPERATION
       TYPE = SPDistDensityToFieldOp
       FIELD_DATA_NAME = FieldData
       SPDIST_DATA_NAME = He-DST
       PSORT_NAME = Sort_He-DST
       FIELD_NAME = NHe CNHe # Computational and Physical Number per Cell
END OPERATION
######################
## Write VTK Output ##
######################
DEFINE OPERATION
       TYPE = LogicalFieldWriteVTKOp
       FIELD_DATA_NAME = FieldData
       FILE_HEAD = heatbath_1/plt_
       FIELD_NAME = NHe CNHe
       SKIP = 5
END OPERATION
```

The operation `LogicalFieldSetOp` sets the value of the field to zero for the given field name; in this case, we have an operation for CNHe and another for NHe. The current version of TURF uses ËFieldDataäs the default name of the field data. The operation `SPDistDensityToFieldOp` sums the quantity of helium for each cell and stores it into CNHe and NHe. The last operation `LogicalFieldWriteVTKOp` outputs the field data for CNHe and NHe in VTK format every five iterations.

Opening the VTK files with visualization software such as VisIt we notice TURF has created a cube with a uniform density of helium particles just as we expected. As the simulation progresses in time, the particles remain unchanged. In the next tutorial TURF Heatbath Example: Part 2 we will discuss how to push particles through the domain and how to handle particles that leave the specified domain.

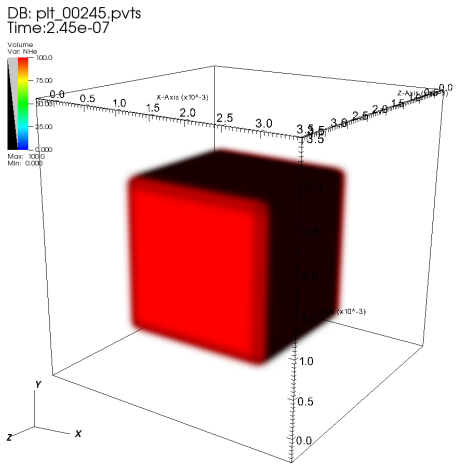


Fig. 2.5: The final state of the helium particles visualized in VisIt.

2.3 Heatbath 2: Evolution

Jonathan Tran

2.3.1 Introduction

This tutorial is the second of two parts which give an overview of the heatbath example in TURF. If you have not yet gone over the first heatbath tutorial, it is advised that be done prior to running this example. You should already have installed TURF and verified that it was installed properly. For information on this please read README located in src-TURF. All relevant files can be located in `tutorial-TURF/TURF-IR_2016/Heatbath/Heatbath_Part2`. You should see several files with the `.list` extension, which act as the scripting files for TURF.

In the first part of the heatbath tutorial we discussed the basics of constructing a world coordinate system, domain and the operations needed to add particles to the simulation. In this tutorial we plan on expanding our simulation by adding a time dependence. Section 2.3.2 discusses how to thermally expand particles and how to handle particles which have left the domain. In Section 2.3.3, we will impose boundary conditions which specularly reflect incoming particles, thus completing the heatbath example. Lastly, we would like to construct the same simulation with multiple domains requiring changes to both the world.list and operations.list files. The list of operations used in this tutorial is provided in Table 2.5.

Table 2.5: Summary of operations used in this tutorial.

Stage	Operation	Description
INITIALIZE	`SPDistConstantIC`	Initial particle distribution
MOVEOP	`NextStageOp`	Continue to next stage
	`SPDistMoveOp`	Advancement of particles
	`SPDistDensityToFieldOp`	Sum particles per cell for field entry
	`SPDistSortOp`	Sort particles according to cell ID
	`LogicalFieldSetOp`	Initialize the field parameters
	`SPDistBCSpecOp`	Specularly reflecting boundary condition
	`SPDistCombineOp`	Unifies the particles from different distributions
	`SPDistCellIDOp`	Marks the cell ID in which particles reside
	`SPDistSplitOp`	Splits particle distribution into two by cell ID
	`SPDistPatchOp`	Transfers particles between domains
POSTOP	`LogicalFieldWriteVTKOp`	Write to output files for 3D plots

2.3.2 Particle Pushing

In TURF Part 1 of Heatbath example, our final example had particles which remained stationary for the length of the simulation. The next logical step is to allow the particles to thermally expand.

operations.push-untrimmed.list

Pushing particles is quite simple, requiring one additional operation. We begin by running the `operations.push-untrimmed.list` file the same way as before. In this case after initially distributing the particles in a cube, the operation `SPDistMoveOp` will thermally expand the particle distribution over time. You may notice the total number of particles in the simulation decreasing over time. This is due to a lack of boundary conditions for our domain; particles continue on their trajectory beyond the bounds of the simulation.

```
DEFINE OPERATION
      TYPE = SPDistMoveOp
      SPDIST_DATA_NAME = He-DST
END OPERATION
```

operations.push.list

Running this example shows an output identical to the push-untrimmed case however, it now has functionality to sort through the particle list and explicitly remove those that have moved into the ghost cells as opposed to the previous example where these particles would be ignored and continue on their trajectory. It does so using the following operations:

```
DEFINE OPERATION
      TYPE = SPDistConstantIC
      SPDIST_DATA_NAME = He-GST
      MAX_NP = 1280000
END OPERATION
################################################
## Initial Sort Removes Particles Outside Domain ##
################################################
DEFINE OPERATION
      TYPE = SPDistCellIDOp
      SPDIST_DATA_NAME = He-DST
END OPERATION
DEFINE OPERATION
      TYPE = SPDistSortOp
      NAME = Sort_He-DST
      SPDIST_SRC_NAME = He-DST
      SPDIST_DST_NAME = He-GST
END OPERATION
```

From the previous example we have added a distribution for helium named He-GST. The operation `SPDistCellIDOp` determines what cell every particle is in and the operation `SPDistSortOp` moves particles in ghost cells from He-DST to He-GST.

2.3.3 Particle Heatbath

operations.heatbath.list

The final example is the particle heatbath, thermally expanding in a box. To do so, we impose boundary conditions and have particles specularly reflect off the walls of the domain. By running `operations.heatbath.list`, we use an operation named `SPDistBCSpecOp` which creates regions that share a surface with the domain. These regions will reflect incoming particles in a given direction. An example of the use of this operation is shown below. The code requires us to write this operation six times, one for every surface of the cubic domain. A visualization of these regions can be seen in Fig. 2.6, and the final state visualized by VisIt is shown in Fig. 2.7. Taking a look at the output in VisIt, we notice the total number of particles remains unchanged. If the simulation is run longer, it will eventually approach steady state.

```
##############################
## 1-Walls reflect particles ##
##############################
DEFINE OPERATION
      TYPE = SPDistBCSpecOp
      SPDIST_DATA_NAME = He-DST
      DIRECTION = xm
      BOUND_LO = (-100.e-4,-100.e-4,-100.e-4)
      BOUND_HI = ( 0.00001e-4, 132.e-4, 132.e-4)
END OPERATION
```

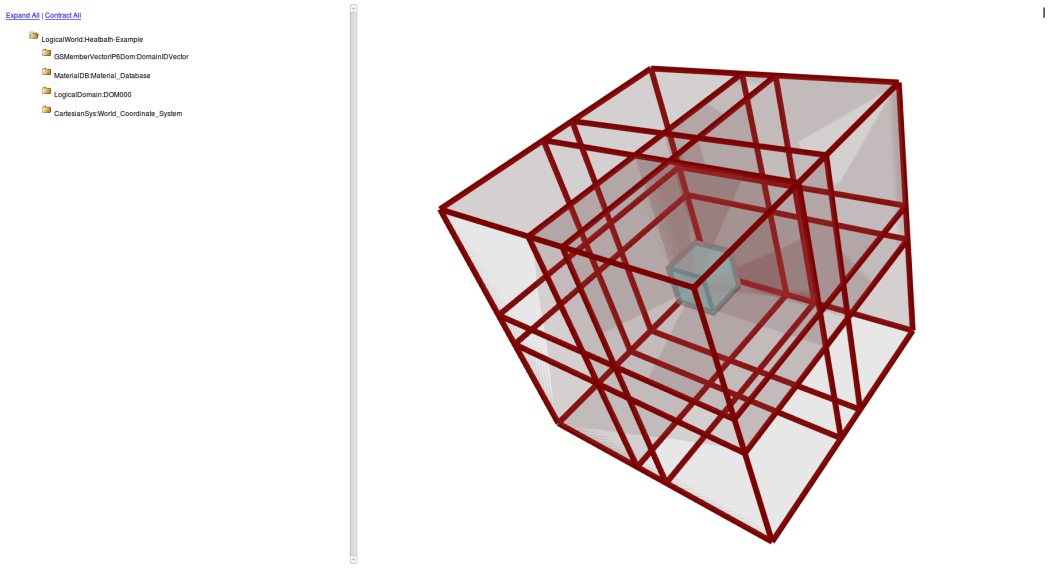


Fig. 2.6: `World-Rank(0).html` visualizing both the domain and boundary condition region which specularly reflects incoming particles.

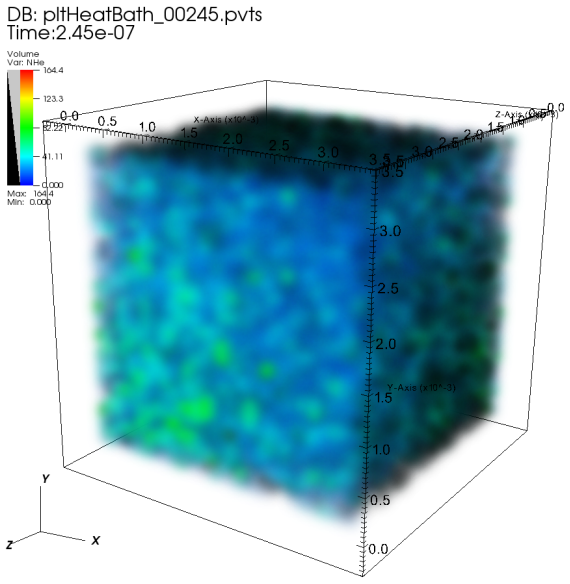


Fig. 2.7: The final state of the helium particles visualized in VisIt for the heatbath example.

Multiple Domain Case

The final example demonstrates the ability to use multiple domains. Doing this requires us to change the domain geometry in the world.list file. Luckily for us we already have a file we can change the pointer to named world.heatbathx2.list. We first remove the previous pointer and create a new pointer with the same name to `world.heatbathx2.list`.

```
tutorial-TURF/TURF-IR_2016/Heatbath/Heatbath_Part2> rm world.list
tutorial-TURF/TURF-IR_2016/Heatbath/Heatbath_Part2> ln -s world.heatbathx2.list world.list
```

Comparing the two world.list files the domain geometry is the only modification.

```
#############################################################################
## Domain Geometry
#############################################################################
DEFINE DOMAIN DOM000
      BOUNT_LO = (0.0,0.0,0.0)
      BOUND_HI = (1.6e-3,3.2e-3,3.2e-3)
END DOMAIN
DEFINE DOMAIN DOM001
      BOUND_LO = (1.6e-3,0.0,0.0)
      BOUND_HI = (3.2e-3,3.2e-3,3.2e-3)
END DOMAIN
```

When using multiple domains, it must be possible to exchange particles between the different domains. Looking at the operations.list file we notice two significant differences between the single domain and multiple domain cases. The first of which handles the exchange of particles from one domain to the other using a distribution named He-EXC. The operation `SPDistCombineOp` unifies the particles from He-EXC distribution with He-DST at the beginning of every loop.

```
DEFINE OPERATION
      TYPE = SPDistConstantIC
      SPDIST_DATA_NAME = He-EXC
      MAX_NP = 1280
END OPERATION
############################################################
## Combine EXC into DST from Patch at End of Move Stage ##
############################################################
DEFINE OPERATION
      TYPE = SPDistCombineOp
      SPDIST_SRC_NAME = He-EXC
      SPDIST_DST_NAME = He-DST
# VERBOSE = TRUE
END OPERATION
```

If a particle moves between the two domains, this particle is marked by assigning the maximum cell index to the particle's cell ID (`SPDistCellIDOp`). Then, the particle is temporarily removed from He-DST and placed into the distribution He-EXC (`SPDistSplitOp`). Finally, the exchange distribution, He-EXC, is patched between the domains (`SPDistPatchOp`), which takes place between the current and next stages.

```
############################################################
## Split Particles Still Outside Active Domain for Patch ##
############################################################
DEFINE OPERATION
      TYPE = SPDistCellIDOp
```

```
      SPDIST_DATA_NAME = He-DST
END OPERATION
DEFINE OPERATION
      TYPE = SPDistSplitOp
      SPDIST_SRC_NAME = He-DST
      SPDIST_DST_NAME = He-EXC
END OPERATION
DEFINE OPERATION
      TYPE = SPDistPatchOp
      SPDIST_SRC_NAME = He-EXC
      SPDIST_DST_NAME = He-EXC
END OPERATION
DEFINE OPERATION
      TYPE = NextStageOp
END OPERATION
```

Running the simulation we see that the output is similar to the single domain heatbath case as we would expect it to be, do note however in the `world.list` file the overlapping ghost cells in the volume domain which are required for the exchange of particles between domains.

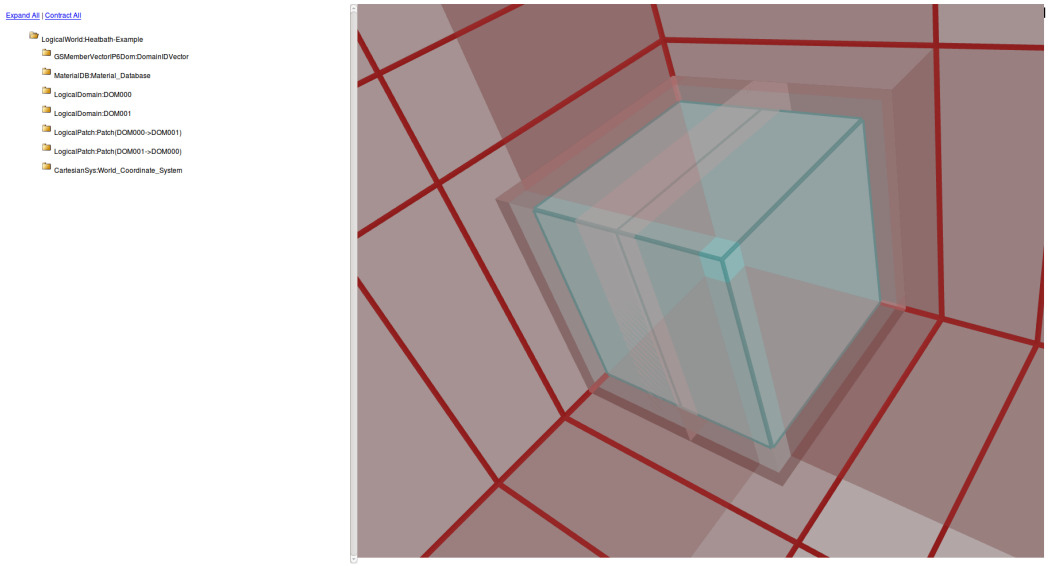


Fig. 2.8: Volume domain split into two pieces.

2.4 Grounded Box: 3D Electrostatic PIC

Robert Martin

2.4.1 Introduction

This tutorial demonstrates running a simple 3D electrostatic particle in cell (PIC) case in the Thermophysics Universal Research Framework (TURF). This tutorial assumes familiarity with the simple heatbath tutorials. New users are referred to those tutorials for further explanation. The TURF input files can be located in `tutorial-TURF/TURF-IR_2016/GroundedBox/ES-PIC`. You should see two files with the `.list` extension, which act as the scripting files for TURF.

The grounded box test case was developed to verify TURF's PIC algorithms with respect to AFRL/RD's ICEPIC particle in cell code running in electrostatic PIC mode [11]. The initial conditions are a uniform unit meter cube of zero velocity protons at a density of 10^{10} m^{-3}. In one octant of the cube, the proton charge is neutralized with 10^{10} m^{-3} electrons initially at stationary. The walls of the cube are set to a fixed 0-Volt potential. The electrons are then accelerated by the field due to the charge of the non-neutralized protons in the remaining 7 octants of the box. The field evolves as the electrons accelerate such that the cloud oscillates and evolves within the box. Particles that hit the edge of the box are assumed to be neutralized and removed from the simulation.

2.4.2 world.list

There are two world files, `world.single.list` and `world.multi.list`, setup for single- and multi-domain simulations respectively. To run a simulation either link one of the existing world files to world.list, e.g. `ln -s world.single.list world.list`, or run TURF with the `-input` option, e.g. `TURF-o -input world.single.list`. For generality `world.list` is used instead of listing both world files. Running the TURF executable in the working directory will have TURF search for the default script file, `world.list`, and parse it automatically. We can take a look at the script by opening the file. The first block that defines the `WORLD` is shown below.

```
DEFINE WORLD
       NAME = ICEPIC-Bench
       OP_FILE = operations.list
       COORDINATES = cartesian
       ORIGIN = (0.0,0.0,0.0)
       DELTA = (0.02,0.02,0.02)
       END_TIME = 10.0e-6
       START_DT = 2.50e-9
       FIELDS = [rho, Enx, Eny, Enz, phi, residual_phi, Np+, Ne-, CNp+, CNe-]
       STAGES = [INITIALIZE, SOLVE, MOVE, PLOT]
END WORLD
```

The options defined in this file should look familiar after completing the heatbath tutorial. In this example, the world is named "ICEPIC-Bench" to denote that it was originally intended to serve as a benchmark verification run against the ICEPIC code. The sample uses the `operations.list` operations file to define the simulation algorithm which will be discussed below. The remainder of the world definition sets a global Cartesian coordinate system with 2 cm cells along with 2.5 ns time steps up to a final simulation time of 10 μs. The next line defines 10 field variables for charge density (`rho`), 3 node centered electric field components (`En`), the electrostatic potential (`phi`), an auxiliary variable for calculating the residual of the potential during the field solve (`residual_phi`), and proton and electron physical and computational particle counts in cells (`Np+,Ne-,CNp+,CNe-`).

The example run is broken into 4 stages named `INITIALIZE`, `SOLVE`, `MOVE`, and `PLOT`. The two additional stages compared to the heatbath example are to accommodate an iterative electrostatic potential solve stage (`SOLVE`) and to ensure synchronization prior to the plotting operation stage (`PLOT`) though the latter is not strictly necessary. In this tutorial, the operations file will be considered in stages (Section 2.4.3). List of operations defined in `operations.list` are given in Table 2.6.

The last section of `world.list` defines the active domain for the simulation. In this example, it is simply a 1 m unit cube starting from the coordinate origin. Using the global mesh spacing of 2 cm from the `WORLD` definition

results in a $50 \times 50 \times 50$ active cell cube with the default 3 "ghost"-cells added to the high and low side in each direction for application of boundary conditions. For a single domain simulation the mesh bounds are defined via,

```
################################################################################
## Domain Geometry
################################################################################
DEFINE DOMAIN DOM000
      BOUND_LO = (0.0,0.0,0.0)
      BOUND_HI = (1.0,1.0,1.0)
END DOMAIN
```

and for the multi domain version

```
DEFINE DOMAIN DOM000
      BOUND_LO = (0.0,0.0,0.0)
      BOUND_HI = (1.0,1.0,1.0)
      SUB_DOMAINS = (2,2,2)
END DOMAIN
```

Table 2.6: Summary of operations listed in `operations.list`.

Stage	Operation	Description
INITIALIZE	`SPDistConstantIC`	Initial particle distribution
	`SPDistCellIDOp`	Flag cell in which particle resides
	`SPDistSortOp`	Sort particles in cells by CellID
	`LogicalFieldSetOp`	Set field values to constant
	`SPDistDensityToFieldOp`	Sum real and computational particles/cell to field
	`SPDistToFieldOp`	Sum particle charges to field entry
	`NextStageOp`	Continue to next stage
SOLVE	`LogicalBCConstantOp`	Set value of cell centers in box every iteration
	`LogicalPoissonStripOp`	Red/Black line relaxing Poisson solve
	`LogicalResidualOp`	Calculate residual of Poisson solve
	`LogicalNormOp`	Calculate L^p-norm of field variable
	`CriteriaStageOp`	Continue to next stage if quantity below criteria
MOVE	`LogicalNodeGradientOp`	Calculate node-centered gradient of cell center field
	`SPDistESPushOp`	Electrostatic particle push using node electric field
	`SPdistSplitOp`	Split particle distribution by CellID flag
PLOT	`VolumeRenderOp`	Single cubic domain realtime volume rendering
	`LogicalFieldWriteVTKOp`	Write to output files for 3D plots

2.4.3 operations.list

Stage: Initialize

This stage creates new particle electron and proton particle distributions. The `SPDistConstantICOp` operation should be familiar from the heatbath example. Notable differences are that the charges, `Z`, are non-zero and the electron mass is defined in units of electron mass instead of proton mass. Here, `FILL_RATIO` is roughly the percentage of particles to fill SPDist class object.

```
DEFINE STAGE INITIALIZE

########################################################################
## Initial Particle Distributions and Ghost/Exchange Distributions ##
########################################################################
DEFINE OPERATION
      TYPE = SPDistConstantICOp
      SPDIST_DATA_NAME = e-DST
      MAX_NP = 2000000
      FILL_RATIO = 1.0
      BOUND_LO = (0.0,0.0,0.0)
      BOUND_HI = (0.5,0.5,0.5)
      TEMPERATURE = 0.0
      NUMBER_DENSITY = 1.e10
      Z = -1
      MASS = 1.0 Me
      VEL = (0.,0.,0.)
END OPERATION

DEFINE OPERATION
      TYPE = SPDistConstantICOp
      SPDIST_DATA_NAME = p+DST
      FILL_RATIO = 0.0625
      MAX_NP = 2000000
      BOUND_LO = (0.0,0.0,0.0)
      BOUND_HI = (1.0,1.0,1.0)
      TEMPERATURE = 0.0
      NUMBER_DENSITY = 1.e10
      Z = 1
      MASS = 1.0 Mp
      VEL = (0.,0.,0.)
END OPERATION
```

Next, empty "ghost" particle distributions are created again using `SPDistConstantICOp`. These are empty buffers where particles that have escaped the domain get copied later on in the sort.

```
DEFINE OPERATION
      TYPE = SPDistConstantICOp
      SPDIST_DATA_NAME = e-GST
      MAX_NP = 2000000
END OPERATION

DEFINE OPERATION
      TYPE = SPDistConstantICOp
      SPDIST_DATA_NAME = p+GST
```

```
      MAX_NP = 2000000
END OPERATION
```

The next set of operations sort the particles by the cell in which they reside. Though still part of the "INITIALIZE" stage, they will run every time the simulation loops back through that stage. The SPDistCellIDOp operation identifies which cell the particle resides in and saves it to the "CellID" variable within the particle distribution. The SPDistSortOp operation sorts the particles by their "CellID" and any particle that has escaped the domain gets separated into the ghost distribution.

```
################################################
## Initial Sort Sets Cell Edges for Fast Sums ##
################################################

DEFINE OPERATION
      TYPE = SPDistCellIDOp
      SPDIST_DATA_NAME = e-DST
END OPERATION

DEFINE OPERATION
      TYPE = SPDistCellIDOp
      SPDIST_DATA_NAME = p+DST
END OPERATION

DEFINE OPERATION
      TYPE = SPDistSortOp
      NAME = Sort_e-DST
      SPDIST_SRC_NAME = e-DST
      SPDIST_DST_NAME = e-GST
END OPERATION

DEFINE OPERATION
      TYPE = SPDistSortOp
      NAME = Sort_p+DST
      SPDIST_SRC_NAME = p+DST
      SPDIST_DST_NAME = p+GST
END OPERATION
```

The next sections accumulates particle quantities into the cell field variables. Before the data can be accumulated, the field variables must be cleared using the LogicalFieldSetOp. In future version of TURF, these operations may be optionally fused, but the separate combination is more general. The actual accumulation of field data is performed by the SPDistDensityToFieldOp and SPDistToFieldOp. The first is used to set diagnostic fields for number of real (Nx) and computational (CNx) particles per cell. It is worth noting that these numbers are both raw sums. To obtain the density "n" from particle count N, LogicalFieldVolumetricMulOp simply divides the particle count by the cell volume. This usage can be seen in the Vlasov collisionless shock tutorial. The second operation multiples by particle charge while doing the accumulation such that the charge density is computed. More computationally efficient calculations can now be performed using field multiplication and summation operations, but the example in this tutorial was created using an early version of TURF that existed before those operations had been completed. This functionality can also be seen in the Vlasov collisionless shock tutorial.

```
###################
## Sum to Fields ##
###################

# Clear Variables First
```

```
DEFINE OPERATION
       TYPE = LogicalFieldSetOp
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = CNe-
       VALUE = 0.0
END OPERATION

DEFINE OPERATION
       TYPE = LogicalFieldSetOp
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = CNp+
       VALUE = 0.0
END OPERATION

DEFINE OPERATION
       TYPE = LogicalFieldSetOp
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = Ne-
       VALUE = 0.0
END OPERATION

DEFINE OPERATION
       TYPE = LogicalFieldSetOp
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = Np+
       VALUE = 0.0
END OPERATION

# Accumulate

DEFINE OPERATION
       TYPE = SPDistDensityToFieldOp
       FIELD_DATA_NAME = FieldData
       SPDIST_DATA_NAME = e-DST
       PSORT_NAME = Sort_e-DST
       FIELD_NAME = Ne- CNe-
END OPERATION

DEFINE OPERATION
       TYPE = SPDistDensityToFieldOp
       FIELD_DATA_NAME = FieldData
       SPDIST_DATA_NAME = p+DST
       PSORT_NAME = Sort_p+DST
       FIELD_NAME = Np+ CNp+
END OPERATION


DEFINE OPERATION
       TYPE = LogicalFieldSetOp
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = rho
```

```
      VALUE = 0.0
END OPERATION

DEFINE OPERATION
      TYPE = SPDistToFieldOp
      FIELD_DATA_NAME = FieldData
      SPDIST_DATA_NAME = e-DST
      FIELD_NAME = rho
END OPERATION

DEFINE OPERATION
      TYPE = SPDistToFieldOp
      FIELD_DATA_NAME = FieldData
      SPDIST_DATA_NAME = p+DST
      FIELD_NAME = rho
END OPERATION
```

Finally, the "INITIALIZE" stage is completed with the NextStageOp operation to proceed to the "SOLVE" stage.

```

DEFINE OPERATION
# Default Criteria to Proceed to the Next Stage
      TYPE = NextStageOp
END OPERATION

END STAGE INITIALIZE

################################################################################
```

Stage: Solve

This stage iterates on solving for the electrostatic potential until the residual is small enough to proceed. The first step of the iterative field solve is to set the boundary condition potential to 0 on all six faces of the box. This is done using the LogicalBCConstantOp operation. The operation is relatively straightforward. The phi variable of the default FieldData object is set to a potential of 0 Volts inside of the box defined by BOUND_LO and BOUND_HI. In this configuration of TURF, the potential is assumed to be cell centered. More specifically, the potential is set to 0 for every cell which has a cell center inside the physically defined box. This may lead to errors on the order of Δx on the location of the application of the boundary condition, but with the boundary conditions defined physically, the solution should converge to the exact solution with grid refinement without manual reconfiguration of the operations. This same approach is used when creating the domains which snap to the nearest approximation of cells based on the physical constraints independent of the underlying mesh resolution. Once again, the NAME variable for the operation is simply a designator label for output readability and the value in the NAME is not evaluated by the code to influence application of the operation. Boundary condition boxes are chosen to be large enough to contain at a minimum the first few layers of cell centers even at the coarsest resolutions run. In regions where the physical boundary conditions overlap, the value will be set repeatedly.

```
################################################################################

DEFINE STAGE SOLVE

DEFINE OPERATION
      TYPE = LogicalBCConstantOp
      NAME = Electrode-X-
```

```
      FIELD_DATA_NAME = FieldData
      FIELD_NAME = phi
      VALUE = 0.0
      BOUND_LO = (-0.1,-0.1,-0.1)
      BOUND_HI = (0.0,1.1,1.1)
END OPERATION

DEFINE OPERATION
      TYPE = LogicalBCConstantOp
      NAME = Electrode-X+
      VALUE = 0.0
      FIELD_DATA_NAME = FieldData
      FIELD_NAME = phi
      BOUND_LO = (1.0,-0.1,-0.1)
      BOUND_HI = (1.1,1.1,1.1)
END OPERATION

DEFINE OPERATION
      TYPE = LogicalBCConstantOp
      NAME = Electrode-Y+
      VALUE = 0.0
      FIELD_DATA_NAME = FieldData
      FIELD_NAME = phi
      BOUND_LO = (-0.1,1.0,-0.1)
      BOUND_HI = (1.1,1.1,1.1)
END OPERATION

DEFINE OPERATION
      TYPE = LogicalBCConstantOp
      NAME = Electrode-Y-
      VALUE = 0.0
      FIELD_DATA_NAME = FieldData
      FIELD_NAME = phi
      BOUND_LO = (-0.1,-0.1,-0.1)
      BOUND_HI = (1.1,0.0,1.1)
END OPERATION

DEFINE OPERATION
      TYPE = LogicalBCConstantOp
      NAME = Electrode-Z+
      VALUE = 0.0
      FIELD_DATA_NAME = FieldData
      FIELD_NAME = phi
      BOUND_LO = (-0.1,-0.1,1.0)
      BOUND_HI = (1.1,1.1,1.1)
END OPERATION

DEFINE OPERATION
      TYPE = LogicalBCConstantOp
      NAME = Electrode-Z-
      VALUE = 0.0
      FIELD_DATA_NAME = FieldData
```

```
      FIELD_NAME = phi
      BOUND_LO = (-0.1,-0.1,-0.1)
      BOUND_HI = (1.1,1.1,0.0)
END OPERATION
```

After the boundary conditions have been set, the actual Poisson solve can be performed. Currently, the set of elliptic solvers in TURF is relatively minimal and includes only red-black Gauss-Seidel and tri-diagonal ADI-type solvers as indicated below by the `LogicalPoissonStripOp` operation. There is also degenerate 1D version of the solver that can be used in fundamentally 1D problems or as an accelerated initial guess for solutions that are primarily one dimensional. The only non-default options for the solver selected were to not cycle sweep directions and to sub-cycle the operation 3 times before continuing. The operation is also applied in a red-black checkerboard in the iterative directions so that the solution is independent of the order in which the line relaxation sweeps are performed.

```
DEFINE OPERATION
      TYPE = LogicalPoissonStripOp
      FIELD_DATA_NAME = FieldData
      FIELD_NAME = phi
      SOURCE_NAME = rho
      MESH_NAME = SMesh
      DIRECTION = 0 # Start with X-sweep
      SUBCYCLE = 3
      DIRCYCLE = FALSE # Default TRUE
END OPERATION
```

The last part of the `SOLVE` stage is defining the criteria to iterate the stage or continue to the next. First, the residual of `phi` is computed in every cell and stored in `residual_phi` using the `LogicalResidualOp`. The `LogicalNormOp` operation calculates the norm of the residual. The `NORM` parameter defines the power p for any L^p-norm. The operation creates a new scalar variable `SUMresidual_phi_L2.00e+00` based off the name of the field in which the residual resides and the power of the norm to store the accumulated total residual. Finally, the `CriteriaStageOp` evaluates whether the summed residual is below the required threshold `CRITERIA`. Each domain applies this operation independently. At the end of each stage, every process collects one vote from every domain as to whether or not to proceed to the next stage or to loop to iterate on the stage. These votes are broadcast across all processes and evaluated by the world when determining whether or not to proceed.

```
DEFINE OPERATION
      TYPE = LogicalResidualOp
      FIELD_NAME = phi
      RESIDUAL_NAME = residual_phi
      SOURCE_NAME = rho
END OPERATION

DEFINE OPERATION
      TYPE = LogicalNormOp
      FIELD_NAME = phi
      RESIDUAL_NAME = residual_phi
      NORM = 2.0
END OPERATION

DEFINE OPERATION
# Threshold Criteria to Proceed to the Next Stage
      TYPE = CriteriaStageOp
```

```
      QUANTITY_NAME = SUMresidual_phi_L2.00e+00
      CRITERIA = 5.0e-4 # High Density!
END OPERATION

END STAGE SOLVE

################################################################################
```

Stage: Move

In the next section, the particle positions are updated using the field solved in the prior step. To do this, the node centered electric field, `En`, is evaluated first using the `LogicalNodeGradientOp` operator. Because the field is the negative gradient of the potential, the `FIELD_MULTIPLY_CONSTANT` of `-1.0` is included. The `FIELD_GRADIENT_DIRECTIONS` are suffixes attached to the root name `En` that the operator uses to construct the three components of the field names needed to store the result.

```
################################################################################

DEFINE STAGE MOVE

DEFINE OPERATION
      TYPE = LogicalNodeGradientOp
      FIELD_DATA_NAME = FieldData
      FIELD_POTENTIAL_NAME = phi
      FIELD_GRADIENT_PREFIX = En
      FIELD_MULTIPLY_CONSTANT = -1.0
      FIELD_GRADIENT_DIRECTIONS = [x, y, z]
END OPERATION
```

The next two operations use the field to advance the electron and ion positions. The inputs are similar to the basic linear push described in the heatbath tutorials with extra field options so that the operator knows which field data to use for the acceleration.

```
DEFINE OPERATION
      TYPE = SPDistESPushOp
      FIELD_DATA_NAME = FieldData
      FIELD_EN_PREFIX = En
      FIELD_EN_DIRECTIONS = [x, y, z]
      SPDIST_DATA_NAME = e-DST
END OPERATION

DEFINE OPERATION
      TYPE = SPDistESPushOp
      FIELD_DATA_NAME = FieldData
      FIELD_EN_PREFIX = En
      FIELD_EN_DIRECTIONS = [x, y, z]
      SPDIST_DATA_NAME = p+DST
END OPERATION
```

After the push, the particle distribution is split between particles that remain within the domain and particles that escaped into the grounded wall. Particle that escape are marked with a `CellID` flag set within the push. This push does not actually test boundary intersections during the push which is a fast method for simple boundary conditions. Triangulated boundary surface intersecting pushes with and without field are still in testing and will appear in future revisions of the TURF-IR. The last operation in the stage is another `NextStageOp`.

```
DEFINE OPERATION
      TYPE = SPDistSplitOp
      SPDIST_SRC_NAME = e-DST
      SPDIST_DST_NAME = e-GST
END OPERATION

DEFINE OPERATION
      TYPE = SPDistSplitOp
      SPDIST_SRC_NAME = p+DST
      SPDIST_DST_NAME = p+GST
END OPERATION

##########################################

DEFINE OPERATION
# Default Criteria to Proceed to the Next Stage
      TYPE = NextStageOp
END OPERATION

END STAGE MOVE
################################################################################
```

Stage: Plot

The last stage of the simulation is plotting. The first operation in the plotting section is the CUDA accelerated real-time volume ray tracing operation, `VolumeRenderOp`. The code is primarily a wrapped version of the NVIDIA CUDA SDK's VolumeRender example. The infrastructure launches that set of code in a separate window. When the operation is applied during the code's main thread loop, a second buffer is filled from the field variable specified by the `FIELD_DATA_NAME` and `FIELD_NAME` parameters. It then signals the visualization thread to swap buffers. It is restricted to single cubic domains in this version of the infrastructure because it uses the rendering kernels from the example with few modifications to apply in other geometries. Most of the settings for producing the coloring and view were obtained by interacting with the visualization to determine a "good" view. This mode of interaction is described below the file listing. Other options include the commented `FILE_HEAD` and `SAVE_IMG` options. If re-enabled, the operation outputs a '.ppm' image file for every iteration that is drawn. Iteration skipping can be adjusted by the `SKIP` parameter to reduce the number of files. The `VIEW_ORBIT` parameter tells the visualization to rotate by the indicated number of degrees once per iteration automatically in addition to the interactive rotations to help make the 3D nature of the volume rendering more intuitive.

```
################################################################################
DEFINE STAGE PLOT

DEFINE OPERATION
      TYPE = VolumeRenderOp
      FIELD_DATA_NAME = FieldData
# FILE_HEAD = VolViz0T
# SAVE_IMG = TRUE
      FIELD_NAME = Ne-
      SKIP = 1
# INVERT = TRUE
      DENSITY = 0.04
      BRIGHTNESS = 1.7
      TRANSFERUPPERBOUND = 3.8e5
```

```
      TRANSFERLOWERBOUND = 2.5e3
      LOG_PLOT = FALSE
      INVERT = FALSE
      VIEW_TRANSLATION = (0.0,0.0,-3.6)
      VIEW_ROTATION = (0.4,51.6,0.0)
      VIEW_ORBIT = (0.0,-2.0,0.0)
      WINDOW_SIZE = (960,960)
END OPERATION
```

An example of the output displayed with the default settings by the realtime visualization can be seen in Fig. 2.9. This shows the electron cloud density in the box after 828 timesteps using the default visualization parameters.

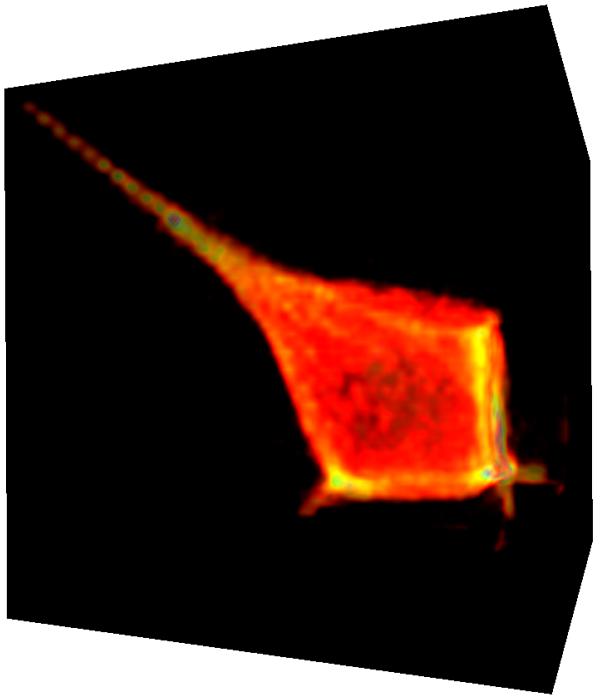


Fig. 2.9: Volume rendering example output of electron density in grounded box

Left-clicking and dragging the mouse rotates the visualization. Right-clicking and dragging the mouse scales the view. Center or simultaneous left and right clicking while dragging the mouse pans the viewport. The '-+' keys adjust the density for the ray tracing. Lower values make the electron cloud more translucent and higher makes the rendering thicker and only values closer to the surface of the cloud are visible. The square bracket keys, '[]', adjust the 'brightness' of the display. The keys on the next row down, ';'', adjust the 'transferUpperBound', which is essentially the top edge of the colormap. The next row down from there, the ',.' keys adjust the 'transferLowerBound'. This is similarly the bottom edge of the colormap. The 'i' key inverts the coloring of the display to a black box on a white background. As the keys adjust the settings, the visualizer displays the adjusted parameters interwoven with normal output from the infrastructure. Once a good view has been determined, the options can then be fed back into the operation's parameters for future runs. The output of holding the '-' is shown below with some additional whitespace for clarity while a similar line is produced by the mouse adjustments as well.

```
Iteration 1747: Time=4.367510e-06 dt=2.500000e-09 [Wall Clock:477.743864]
```

```
density = 0.07, brightness = 2.10, transferUpperBound = 3.45e+05,
       transferLowerBound = 1.39e+04, invert = F
density = 0.06, brightness = 2.10, transferUpperBound = 3.45e+05,
       transferLowerBound = 1.39e+04, invert = F
density = 0.05, brightness = 2.10, transferUpperBound = 3.45e+05,
       transferLowerBound = 1.39e+04, invert = F
NORM: 4.386121e-04
Iteration 1748: Time=4.370010e-06 dt=2.500000e-09 [Wall Clock:478.055719]
```

The last additional operations are a commented version of the `LogicalFieldWriteVTKOp` which writes the field data to output files rather than relying on the realtime visualization. This is necessary for running the tutorial on systems that do not include NVIDIA GPU's that are compatible with the direct OpenGL interface used by the volume renderer. The options are similar to those of the heatbath tutorial. The last operation is a final `NextStageOp` to tell the code to advance to the next stage, or in this case, loop back to the first stage.

```
# DEFINE OPERATION
# TYPE = LogicalFieldWriteVTKOp
# FIELD_DATA_NAME = FieldData
# FILE_HEAD = plt/plt_
# FIELD_NAME = Ne- CNe- Np+ CNp+ phi rho
# SKIP = 5
# # FORMAT = BINARY # Won't open!
# HELP = TRUE
# END OPERATION


DEFINE OPERATION
# Default Criteria to Proceed to the Next Stage
      TYPE = NextStageOp
END OPERATION

END STAGE PLOT
################################################################################
```

2.4.4 Results

The example in this tutorial was originally developed to verify TURF functionality with respect to the ICEPIC code. Using as similar parameters as possible between the two codes, the example was run and visualized in ParaView as shown in Fig. 2.10. The setup was nearly identical to what was outlined above except more particles were used to provide smoother output. In particular, a `FILL_RATIO` of `4.0` was used with the `e-DST`, and the default value of `0.5` was used for the `p+DST` to ensure a similar number of particles were used in TURF as in ICEPIC. For the realtime visualization, the low proton numbers make little difference in the electron density visualization, but they make charge density plots like those used to compare the code much more noisy. The agreement between the two codes was very reasonable considering all the particle trajectories are coupled to the field solution and vice versa. A major difference is the appearance of more charge neutrality on the surface of the ICEPIC result, but this is essentially a difference due to node-centered versus cell-centered output between the two codes. The background in ICEPIC is slightly noisier as well because the ratio of real to computational weights of particles in TURF are modified to ensure the intended cell densities rather than randomly inserting equal weight particles throughout the domain. On longer timescales after the protons have had the opportunity to move further, the noise level in TURF would appear more similar to that in the ICEPIC result. The `SPDistBoxICOp` available in the `TURF-DEV` development package would provide a more directly comparable initialization with constant particle weights, but

it was added after the original verification runs were performed and will not be included in the infrastructure core until the next revision.

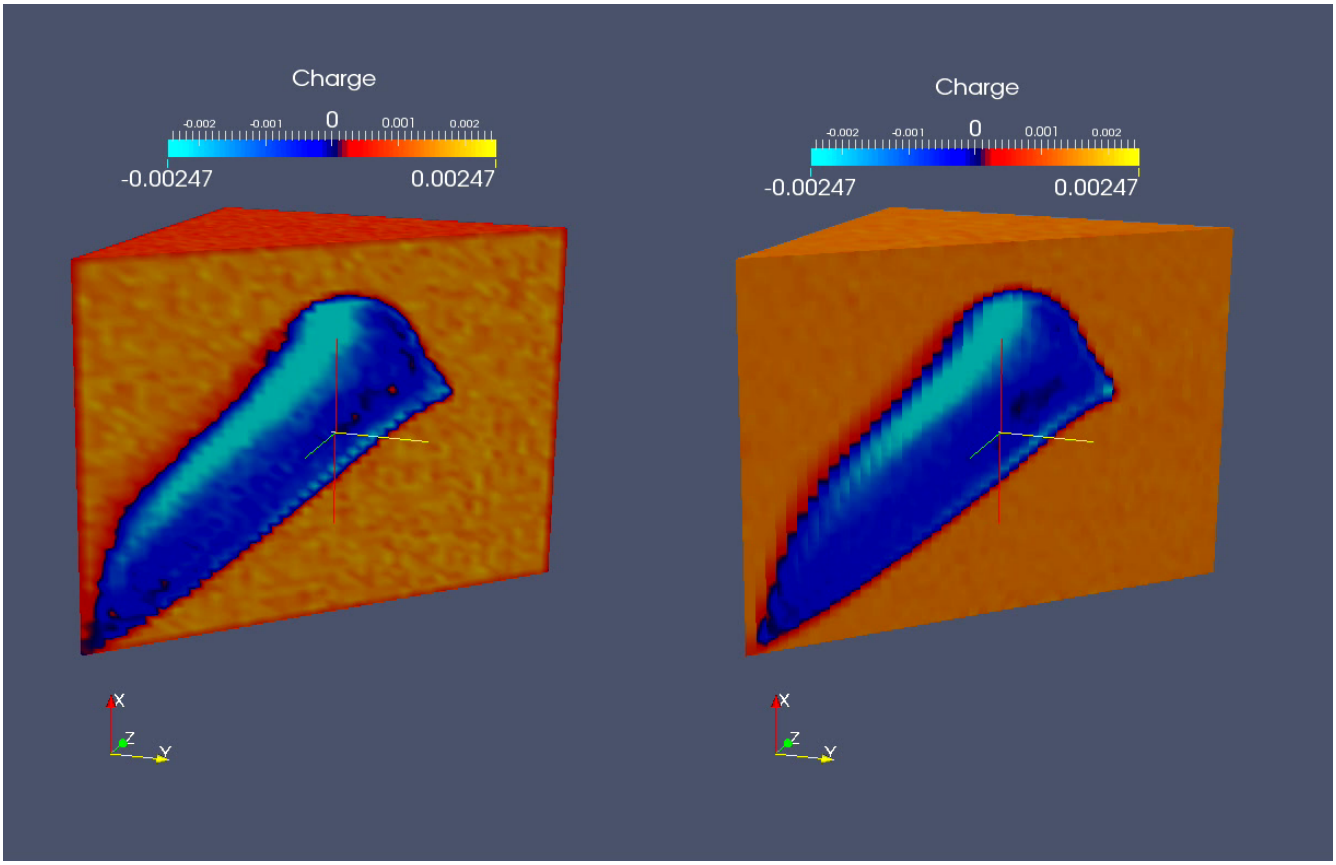


Fig. 2.10: Comparison of ICEPIC (left) and TURF (right) grounded box results

2.5 One Dimensional Normal Shock: DSMC

Samuel J. Araki

2.5.1 Introduction

This tutorial provides steps in setting up the DSMC example provided in the following directory: `tutorial-TURF/TURF-IR_2016/1DShock/DSMC/TURF`. This folder contains two subfolders, `DS1V` and `TURF`. The `DS1V` contains reference cases for Argon shocks using the DS1V code described in Section 2.5.5 below. The `TURF` folder contains input files to run the same Argon shock cases within TURF. For example, the `TURF/M1.2` folder contains the input files for running the Mach 1.2 shock test case. Once in the specific case directory, `world.list` should be pointed to the file `world.dsmc1Dshock.list` by typing

```
tutorial-TURF/TURF-IR_2016/1DShock/DSMC/TURF> ln -s world.dsmc1Dshock.list world.list
```

in the command line, as in other examples. The script file `world.dsmc1Dshock.list` will execute the DSMC operations listed in `operations.dsmc1Dshock.list` to simulate the 1D normal shock problem using the DSMC method. In order to run DSMC simulations properly, the value for `FNUM` must be set to the same value for all the DSMC operations defined in `operations.dsmc1Dshock.list`, where `FNUM` is the number of physical particles represented by a single simulation particle. Therefore, when introducing particles into the domain, operations that require the user specified `FNUM` must be used; these operations are named as `SPDistDSMCConstantICOp` and `SPDistDSMCConstantBCOp`. The operation `SPDistDSMCConstantICOp` distributes particles uniformly inside the simulation domain. On the other hand, the operation `SPDistDSMCConstantBCOp` creates particles outside the domain, and a fraction of these particles flow into the domain. The DSMC collision calculation is done in `SPDistDSMCOp`, and the same `FNUM` used in the other operations must be used. Furthermore, `SPDistDSMCSampleOp` triggers the mixing of the field values between iterations, allowing a smoother distribution at the end of simulation. These DSMC operations are explained in this tutorial.

2.5.2 Description of the Example Problem

In a fluid, disturbance information is communicated within a medium at the speed of sound, allowing the upstream flow field to adjust accordingly. However, when the flow velocity is greater then the speed of sound, the disturbance information cannot be communicated fast enough, resulting in a formation of a shock. The shock creates a "discontinuity" or a sudden change in flow properties such as velocity, pressure, and temperature. Across a shock, the pressure and temperature always increase while the velocity always decreases from upstream to downstream. The example to simulate with the DSMC part of TURF is the 1D normal shock problem, in which the shock forms in a plane perpendicular to the flow direction. In this problem, the flow properties at upstream and downstream regions with respect to the shock location are related through the following equations [12].

$$\begin{aligned} \rho_1 u_1 &= \rho_2 u_2 \\ p_1 + \rho_1 u_1^2 &= p_2 + \rho_2 u_2^2 \\ h_1 + \frac{1}{2}u_1^2 &= h_2 + \frac{1}{2}u_2^2 \end{aligned} \tag{2.1}$$

where ρ is the density, u is the velocity, p is the pressure, h is the enthalpy, and subscripts 1 and 2 denote upstream and downstream, respectively. Equation (2.1) is obtained by integrating the Euler equations, a set of conservation equations for mass, momentum, and energy that are applicable for such flows [13]. In a perfect gas, the speed of sound, a, can be determined using the isentropic relation.

$$a^2 = \left(\frac{\partial p}{\partial \rho}\right)_s = \frac{\gamma p}{\rho} = \gamma R_s T \tag{2.2}$$

where γ is the heat capacity ratio defined as $\gamma = 1 + 2/f$, f is the degree of freedom (i.e. 3 for a monatomic gas and 5 for a diatomic gas), R_s is the specific gas constant (i.e. 208.13 J/kg·K for argon), and T is the temperature. Using Eq. (2.1) and the perfect gas assumption, the downstream flow properties can be determined if the upstream

flow properties are known [12].

$$M_2^2 = \frac{1 + \frac{\gamma-1}{2} M_1^2}{\gamma M_1^2 - \frac{\gamma-1}{2}} \tag{2.3}$$

$$\frac{n_2}{n_1} = \frac{\rho_2}{\rho_1} = \frac{u_1}{u_2} = \frac{(\gamma+1)M_1^2}{(\gamma-1)M_1^2 + 2} \tag{2.4}$$

$$\frac{T_2}{T_1} = 1 + \frac{2(\gamma-1)}{(\gamma+1)^2} \frac{\gamma M_1^2 + 1}{M_1^2} \left(M_1^2 - 1\right) \tag{2.5}$$

where M is the Mach number defined as $M = u/a$ and n is the number density. In setting up the 1D normal shock problem, the downstream flow properties need to be evaluated and input in `operations.list` prior to running TURF.

Table 2.7: Downstream flow properties for upstream Mach number of 1.2, 1.4, 2.0, and 8.0. The values are for argon gas.

Downstream			Upstream Mach Number, M_1			
Flow Property	Symbol	Unit	1.2	1.4	2.0	8.0
Velocity	u_2	m/s	294.9	282.3	278.9	667.3
Speed of Sound	a_2	m/s	348.4	376.0	459.4	1456
Mach Number	M_2	-	0.85	0.75	0.61	0.46
Number Density	n_2	$1/m^3$	1.30×10^{22}	1.58×10^{22}	2.29×10^{22}	3.82×10^{22}
Temperature	T_2	K	350.1	407.8	608.9	6116

2.5.3 Setting up the DSMC Example

One way to set up the 1D normal shock problem is to introduce uniformly distributed gases upstream and downstream of the shock location. Given the upstream flow properties, appropriate downstream flow properties are determined by Eqs. (2.3) to (2.5). Table 2.7 provides the downstream flow properties for argon gases of $T_1 = 293$, $n_1 = 1 \times 10^{22}$ m^{-3}, and $a_1 = 318.8$ m/s at M_1 of 1.2, 1.4, 2.0, and 8.0. The upstream flow velocities corresponding to the Mach number of 1.2, 1.4, 2.0, and 8.0 are 382.4, 446.2, 637.4, and 2549.6 m/s, respectively. In order to maintain the gas density and the shock location, the gas should also be flowing into the domain from the upstream boundary according to the flow 1. At the interface between the two gases at different flow properties, the properties are initially discontinuous, while they will develop smooth profiles as time evolves. These profiles can be compared with the profiles obtained by other DSMC models or fluid models. Examples of shock profiles are also provided in Ref. [14].

The script file `world.dsmc1Dshock.list` includes important parameters that define the problem, including the information related to computational grid, time-step, species, etc, as shown below. Referring to `world.dsmc1Dshock.list`, the number of interior cells can be found by dividing (`BOUND_HI-BOUND_LO`) by `DELTA` for each direction. In this example, the grid contains 1000 cells in x-direction and a single cell for both the y- and z-directions. The gas species is argon. Furthermore, the parameters to be output are the number of computational and physical particles which are specified as `NAr` and `CNAr`, respectively. This example uses three stages: INITIALIZE, MOVE, and POSTOP. The script file `operations.dsmc1Dshock.list` contains all the operations within each of the three stages, as listed in Table 2.8. This tutorial only covers the DSMC operations including `SPDistDSMCConstantICOp`, `SPDistDSMCConstantBCOp`, `SPDistDSMCOp`, and `SPDistDSMCSampleOp`. Descriptions of the other operations can be found in other tutorials.

```
DEFINE WORLD
       NAME = DSMC_example
       OP_FILE = operations.dsmc1Dshock.list
       COORDINATES = cartesian
```

```
      ORIGIN = (0.0,0.0,0.0)
      DELTA = (2.0e-5,2.0e-3,2.0e-3)
      END_TIME = 1.0e-4
      START_DT = 1.0e-8
      FIELDS = [NAr, CNAr]
      SPECIES = [Ar]
      STAGES = [INITIALIZE, MOVE, POSTOP]
      START_ITERATION = 0 # Number of Poisson Iteration Before Start
END WORLD

#############################################################################
## Domain Geometry
#############################################################################
DEFINE DOMAIN DOM000
      BOUND_LO = (0.0,0.0,0.0)
      BOUND_HI = (2.0e-2,2.0e-3,2.0e-3)
END DOMAIN
```

Table 2.8: Summary of operations listed in `operations.dsmc1Dshock.list`.

Stage	Operation	Description
INITIALIZE	`SPDistDSMCConstantICOp`	Initial distribution of particles inside the domain
	`SPDistConstantICOp`	Null particle distribution in ghost cells
MOVEOP	`SPDistDSMCConstantBCOp`	Injection of particles
	`SPDistMoveOp`	Advancement of particles
	`SPDistBCSpecOp`	Specular boundary condition
	`SPDistCellIDOp`	Find cell ID associated with particle location
	`SPDistSortOp`	Sort particles according to cell ID
	`SPDistDSMCOp`	DSMC collision calculation
	`LogicalFieldSetOp`	Initialize the field parameters
	`SPDistDensityToFieldOp`	Sum real and computational particles/cell to field
	`LogicalFieldVolumetricMulOp`	Multiplies or divides field data by cell volumes
	`SPDistDSMCSample2Op`	Blend running average of field data
POSTOP	`LogicalFieldWriteVTKOp`	Write to output files for 3D plots
	`LogicalFieldWrite1DOp`	Write to output files for line plots

2.5.4 DSMC Operations

SPDistDSMCConstantICOp

This operation sets up uniformly distributed particles within a box placed inside the simulation domain. An example of inputs for `SPDistDSMCConstantICOp` are shown below. Unlike `SPDistConstantICOp`, the real to computational particle wieght ratio, `FNUM`, is specified as an input. The box to be filled with particles is bounded by `BOUND_LO` and `BOUND_HI`, in which the lower and higher bounds in Cartesian coordinate are specified, respectively. The gas species is argon, and the corresponding mass of each molecule is 39.659 times the proton mass, `Mp`. In order to set up the 1D shock problem properly, the upstream and downstream regions inside the computational domain are filled with particles according to flow properties 1 and 2. The initial particle distribution obtained by the DSMC example is shown in Fig. 2.11. This example corresponds to the case with $M_1 = 1.2$, where the downstream flow properties are given in Table 2.7. Note that `NAr` is related to the number density n such that $n =$`NAr`$/\Delta V$ where ΔV is the size

of a cell in m^{-3}. There is a statistical noise associated with the number of computational particles as the particle locations are determined using the random number generator; the noise can be reduced by increasing the number of simulation particles.

```
DEFINE OPERATION
       TYPE = SPDistDSMCConstantICOp
       SPDIST_DATA_NAME = Ar-DST
       MAX_NP = 12800000
       BOUND_LO = (0.0,0.0,0.0)
       BOUND_HI = (1.0e-2,2.0e-3,2.0e-3)
       TEMPERATURE = 293.0
       Z = 0
       MASS = 39.659 Mp
       NUMBER_DENSITY = 1.0e22
       FNUM = 9.1892e8
       VEL = (382.447,0.0,0.0)
END OPERATION
```

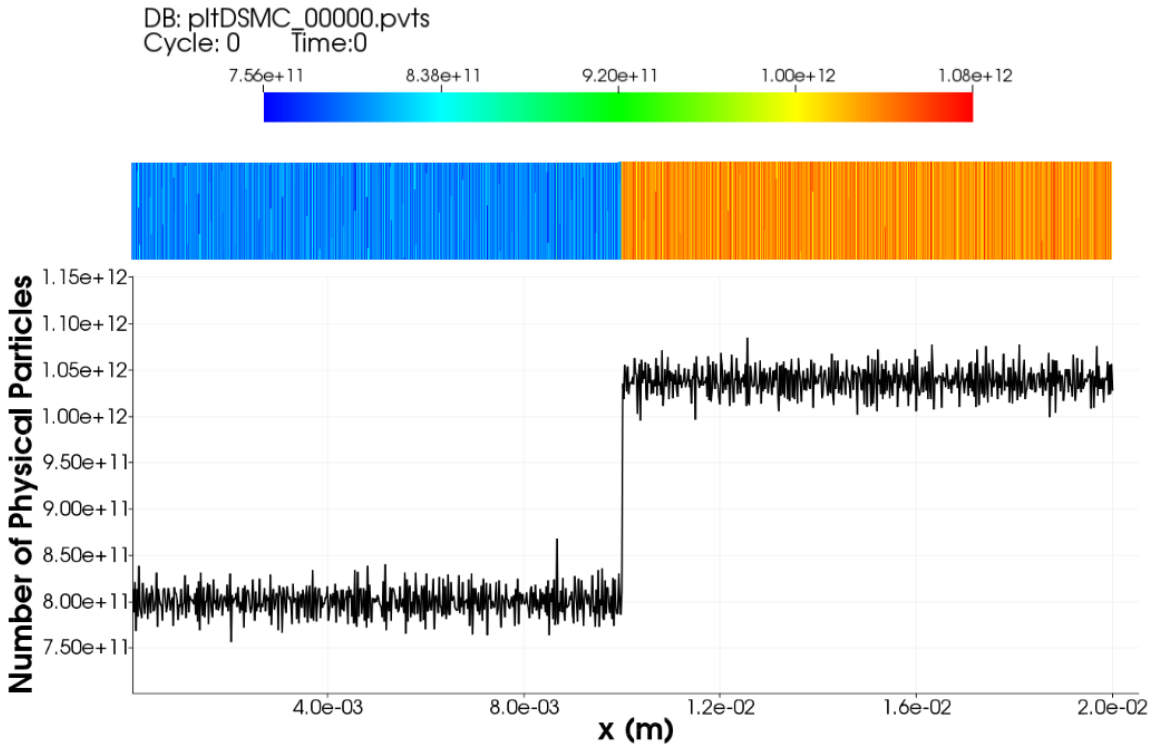


Fig. 2.11: Initial particle distribution.

SPDistDSMCConstantBCOp

This operation sets up uniformly distributed particles within a box placed outside the simulation domain. The size of box is modified such that it lies within the ghost cell region. An example of inputs for `SPDistDSMCConstantBCOp` are shown below. Similar to `SPDistDSMCConstantICOp`, the box to be filled with particles is bounded by `BOUND_LO` and `BOUND_HI`.

In the DSMC example, particles flowing out the simulation domain from $\pm x$ boundaries are simply discarded, while fraction of particles created in the box flows into the simulation domain. When the simulation is at steady-state, the particle counts flowing in and out the domain are maintained to be nearly equal.

```
DEFINE OPERATION
      TYPE = SPDistDSMCConstantBCOp
      SPDIST_DATA_NAME = Ar-DST
      TEMPERATURE = 293.0
      NUMBER_DENSITY = 1.0E22
      FNUM = 9.1892e8
      VEL = (382.447,0.0,0.0)
      BOUND_LO = (-0.01e-2, 0.0e-3, 0.0e-3)
      BOUND_HI = ( 0.00e-2, 2.0e-3, 2.0e-3)
END OPERATION
```

SPDistDSMCOp

This operation finds the number of collisions to perform within all the grid cells and apply collisions based on the DSMC method. The variable hard sphere (VHS) molecular model is used to determine the deflection of particles which requires inputs of `ALPHA` and `DIAM` where `ALPHA` and `DIAM` are the empirical factors that determine the diameter variation and reference diameter of the molecule, respectively. The `DIAM` is the reference molecular diameter at $273K$ such that a hard sphere of that diameter would have the correct fluid viscosity and `ALPHA` is viscosity-temperature power law coefficient, ω, minus the hard sphere value of 0.5. It is important to note that `ALPHA` should not be confused with the variable soft sphere (VSS) model's α parameter. These parameters can be found in Ref. [14].

```
DEFINE OPERATION
      TYPE = SPDistDSMCOp
      SPDIST_DATA_NAME = Ar-DST
      MASS = 39.659 Mp
      ALPHA = 0.31
      FNUM = 9.1892e8
      KOVERM = 208.132
      DIAM = 4.17e-10
      FREQUENCY_TO_RESAMPLE_MFS = 2000
      SORT_OP_NAME = Sort_Ar-DST
END OPERATION
```

SPDistDSMCSampleOp

This operation is similar to `SPDistDensityToFieldOp` except that it starts to mix the fields after the time specified as an input. An example of inputs for `SPDistDSMCSampleOp` is shown below. In the DSMC example, the fields `NAr` and `CNAr` are computed from the particle distribution at the computational grid. After the time given to `MIX_START_TIME`, mixing between iterations is initiated. `MIX_START_TIME` should be set to the time when the simulation becomes steady-state. This operation allows a smooth field distribution at the end of simulation without using a very large number of simulation particles. The field distributions at 2,000 and 10,000 time-steps are shown in Figs. 2.12 and 2.13, respectively. In this example, mixing has been performed after 8,000 time-steps; the distribution is smoothed out significantly after mixing the field parameters for the last 2,000 iterations.

```
DEFINE OPERATION
      TYPE = SPDistDSMCSampleOp
      FIELD_DATA_NAME = FieldData
```

```
      SPDIST_DATA_NAME = Ar-DST
      PSORT_NAME = Sort_Ar-DST
      FIELD_NAME = NAr CNAr
      MIX_START_TIME = 8.0e-5
END OPERATION
```

SPDistDSMCSample2Op

This operation is another version of `SPDistDSMCSampleOp` needed for long sampling times. Because TURF uses single precision floating point numbers for particles and fields for the sake of GPU performance, the direct DSMC sampling starts to loose accuracy for large numbers of samples. For the direct sample operation, the value of the sampled average field is first scaled by $(n_{\text{sample}}-1)/n_{\text{sample}}$ and then particle weights are accumulated into the sample average scaled by w_p/n_{sample}. This means that in each cell, a number of order $(particles/cell)*n_{\text{sample}}$ smaller than the total is added for each particle during the accumulation phase. At approximately $O(1,000-particles/cell)$ and $O(10,000)$ samples, the added pieces are $O(1e7)$ times smaller than the total resulting in lost single precision digits and bulk fluctuation of the sample values. Instead, the `SPDistDSMCSample2Op` uses two density buffers to help alleviate this issue, `NAr` and `NbarAr` (along with particle/cell counterparts). The instantaneous density is first accumulated in `NAr` using the standard `SPDistDensityToFieldOp` operation, and then the sample averaged value is updated via $\bar{N}_{\text{Ar}} = (N_{\text{Ar}} + (n_{\text{sample}} - 1)\bar{N}_{\text{Ar}})/n_{\text{sample}}$ for each cell. Furthermore, all calculations on the right hand side are performed in double precision prior to rounding to attempt to help retain as many digits of precision as possible during the calculation which is impossible with the sum updated in memory per particle as in the original version. The example shown also demostrates the use of the `LogicalFieldVolumetricMulOp` to divide the value of `NAr` by the cell volume to convert absolute number of real particles per cell into number density. These modifications are not necessary for the simple tutorial versions of the shock case because of the minimal sampling performed, but help in converging the results to the DS1V solution as part of the code verification process in the next section.

```
DEFINE OPERATION
      TYPE = SPDistDensityToFieldOp
      FIELD_DATA_NAME = FieldData
      SPDIST_DATA_NAME = Ar-DST
      FIELD_NAME = NAr CNAr # Computational and Physical Number per cell
END OPERATION

DEFINE OPERATION
      TYPE = LogicalFieldVolumetricMulOp
      FIELD_DATA_NAME = FieldData
      FIELD_NAME = NAr
      OP_OPTION = DIV # Divide by Volume (Default is Multiply)
      RUN_AT_INIT = true
END OPERATION

DEFINE OPERATION
      TYPE = SPDistDSMCSample2Op
      FIELD_DATA_NAME = FieldData
      SPDIST_DATA_NAME = Ar-DST
      PSORT_NAME = Sort_Ar-DST
      SRC_FIELD_NAME = NAr CNAr # Instantaneous Computational and Physical Number/Cell
      DST_FIELD_NAME = NbarAr CNbarAr # Sampled Average Computational and Physical Number/Cell
      MIX_START_TIME = 8.0e-5
      SKIP = 5
END OPERATION
```

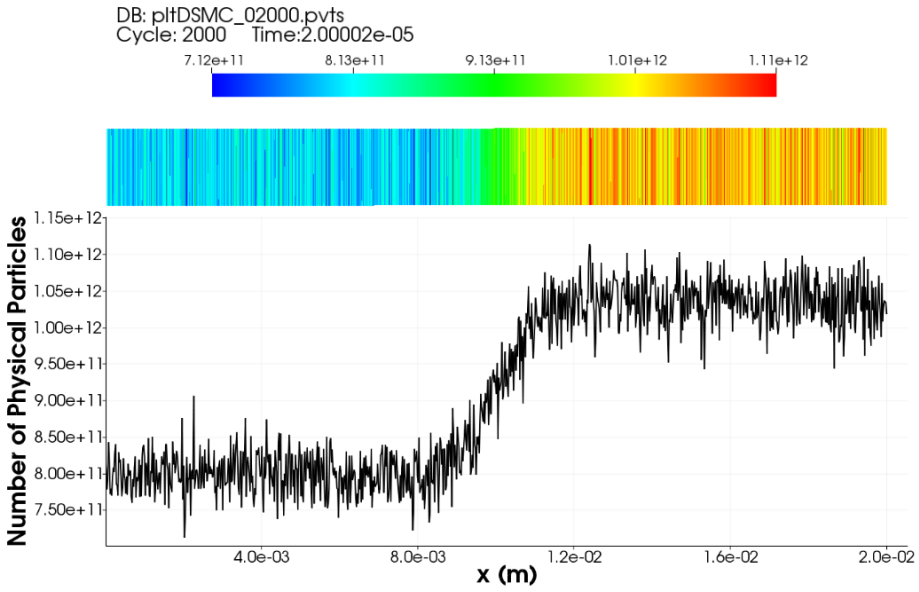


Fig. 2.12: Particle distribution after 2,000 steps.

2.5.5 Comparison of Shock Profiles with Bird's DSMC Code

The 1D shock profiles from the DSMC part of TURF can be compared with other DSMC programs for code verification; in this tutorial, the results are compared with the 1D DSMC code, DS1V, developed by G. A. Bird (available at www.gab.com.au). The DS1V code along with input files (ds1vd.dat) for the cases with M_1 =1.2, 1.4, 2.0, and 8.0 are also provided in this tutorial. In order to obtain a smooth distribution at the end of simulaiton, DS1V is run twice; first, the simulation is started using the "new run" (#3) option in the terminal, the simulation is then stopped at a time greater than 2×10^{-5} sec, and finally the simulation is restarted using "new sample" (#2) followed by the "adapt the cells" (#1) option. The resulting profile of density as a function of position can be extracted from the output file, "PROFILE.DAT."

Figure 2.14 compares the shock profiles computed with TURF and DS1V for the upstream Mach number of 1.2 and 2.0. The values are normalized according to,

$$\tilde{\rho} = \frac{\rho - \rho_1}{\rho_2 - \rho_1} \tag{2.6}$$

The original profile obtained by TURF fluctuated considerably compared to the profile by DS1V when time averaged only over the last $20\mu s$ as shown in Figure 2.12. With the modification to the sampling procedure for $720\mu s$ of time averaging (similar to DS1V) as described in Section 2.5.4, Figure 2.14 show the agreement between the two programs is quite satisfactory.

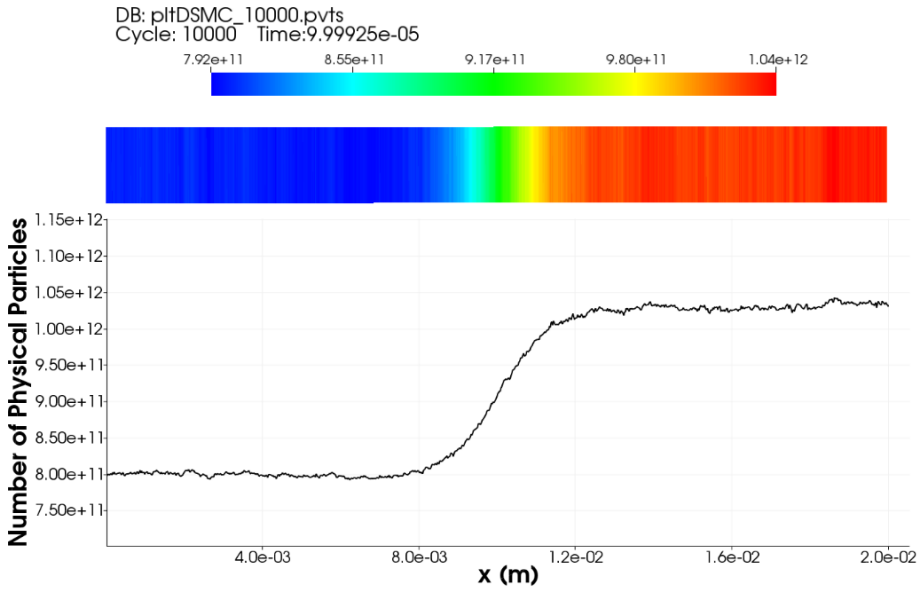
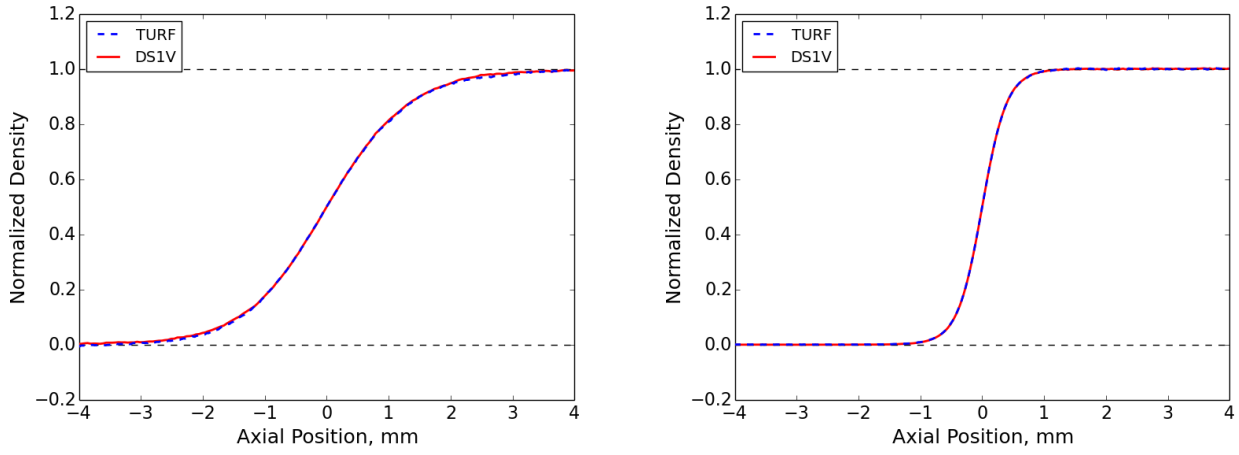


Fig. 2.13: Particle distribution after 10,000 steps.



(a) M=1.2 (b) M=2.0

Fig. 2.14: Normalized density computed by TURF and DS1V.

2.6 Collisionless Electrostatic Shock 1: Experimental Setup

David Bilyeu

2.6.1 Introduction

The purpose of this tutorial is to provide information on the setup of the collisionless electrostatic shock test case. By this point it is assumed that you already have an basic understanding of how the operators in TURF work and the purpose of both the `world.list` and the various `operations.*.list` files. For a review of the purpose of these files please refer to the heatbath example tutorials. Instead this tutorial will focus on aspects that are specific to this simulation and will only provide a cursory overview of basic topics. This tutorial is divided up into three main sections, Section 2.6.2 provides an overview of the collisionless electrostatic shock experiment.

2.6.2 The Collisionless Electrostatic Shock

The collisionless electrostatic shock simulation is based on the experiment by Taylor et.al [15]. Figure 2.15 shows the experimental setup as planned for upcoming validation experiments at AFRL/RQRS including a pulse shape needed to drive a solution used a related problem also originally performed in the same experimental device [16]. In this experiment, Argon gas is fully ionized and separated into driver and target sections of a vacuum chamber. The separation is maintained by a negatively biased grid held at a fixed potential. The number density of the driver side is higher then the target but they share the same ion and electron temperature. At the start of the experiment a ramp potential is applied to the driver side and a shock moves into the target side. For all cases the ion temperature and driven number density was held at 0.2 eV and 10^9 cm^{-3} respectively. A sweep of parameter space was accomplished by varying the electron temperature and the density of the driver gas. The electron to ion temperature varies between 6 and 20 while the density varies from 1 to 20 percent. The initial setup that saw the most study was at a density jump of 25 percent and an electron temperature of 7.5 and 15 eV.

There are several favorable parameters unique to this experiment that makes it an ideal test-case for a Vlasov-Poisson simulation.

- Ion-Ion collisions can be neglected because the mean-free-path between collisions is on the order of 300 Debye lengths (λ_D) and the total domain of the chamber is about $1000\lambda_D$.
- There are no applied magnetic fields and induced currents can be assumed to be negligible which obviate the need to solve the full Maxwell equations.
- The spatial symmetries of the experiment limit variations to be in only one direction so that a 1D1V Vlasov simulation of the flow is sufficient.

In addition to these parameters, several additional assumptions are made which will be tested throughout the upcoming TURF code validation and verification campaign.

- Modeling ion kinetic effects is important due to the collisionless nature of the plasma. This is to be validated through comparison of kinetic and fluid solutions.
- The electrons can be accurately modeled as a Boltzmann equilibrium fluid so that the fastest time scale that needs to be resolved is the ion plasma frequency. This will be tested through comparison with experimental results as well as future fully kinetic simulations which include electron kinetic effects.
- The flow is robust against spontaneously generated transverse modes so that the one-dimensional character of the flow is preserved far enough from the boundaries.

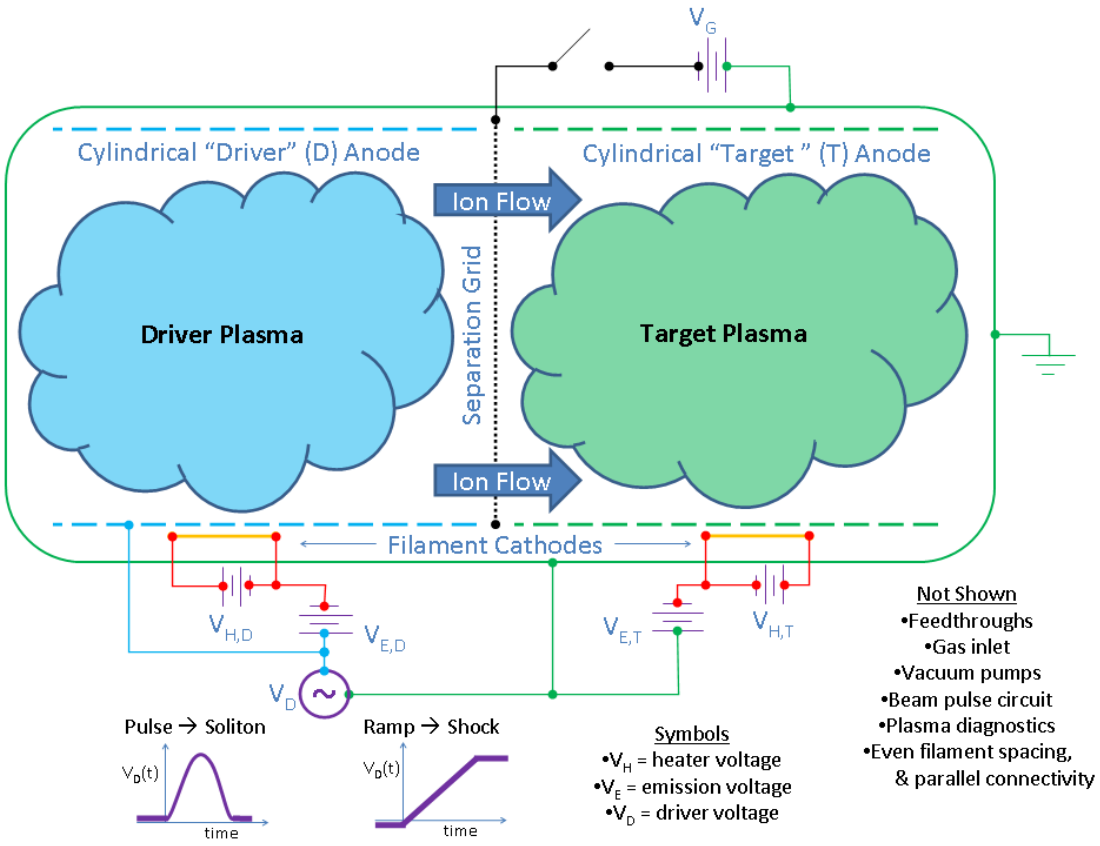


Fig. 2.15: Design of AFRL/RQRS double plasma device experiments.

2.7 Collisionless Electrostatic Shock 2: Vlasov

David Bilyeu

2.7.1 Introduction

The purpose of this tutorial is to provide an overview of how to setup and run a Vlasov simulation in the Thermophysics Universal Research Framework (TURF). By this point it is assumed that you already have an basic understanding of how the operators in TURF work and the purpose of both the `world.list` and the various `operations.*.list` files. For a review of the purpose of these files please refer to the heatbath example tutorials. Instead this tutorial will focus on aspects that are specific to this simulation and will only provide a cursory overview of basic topics. This tutorial is divided up into three main sections, Part 1 of this tutorial provides an overview of the collisionless electrostatic shock experiment, Section 2.7.2 provides an explanation of the `world.list` file, and finally Section 2.7.3 details the operators necessary to run the collisionless electrostatic shock simulation with the Vlasov solver within TURF.

2.7.2 World.list

The `world.list` file defines important parameters that are not unique to any one particular operator. This includes information about the mesh, the time step and total run time, as well as the global variables. The only new information included in the `world.list` that is unique to Vlasov solvers is the definition of the velocity space origin and mesh spacing. These variables are defined via:

```
VELOCITY_ORIGIN = (0.0,-0.5,-0.5)
VELOCITY_DELTA = (15.625,1.0,1.0)
```

At this time each species can have its own unique bounds in velocity space but they must share the same mesh spacing and origin.

This simulation uses seven different field variables including the electron and ion densities, the electric field vector, the electric potential, a density for the electric field solver, mean Velocity, and temperature and are defined:

```
FIELDS = [rhoE, rhoI, Ex, Ey, Ez, phi, rho_source]
FIELDS = [Vmeanx, Vmeany, Vmeanz, Temperature]
```

Furthermore, this example uses three stages, `INIT`, `MOVE`, and `POSTOP` which are responsible for the initialization, solvers, and plotting, respectively. The operations used in each of the three stages are listed in Table 2.9.

2.7.3 operations.vlasov.list

This section explains how to solve this problem using Vlasov methods within TURF. It is assumed that the reader has a basic understanding of how to setup a simulation in TURF and only the information relevant to Vlasov and this simulation in particular are detailed. The operations.vlasov.list file are broken up into three stages just as defined in `world.list`.

Stage: Initialization

In the initialize stage it is necessary to set the initial conditions which includes defining a new phase-space variable. The phase space variable, `fAR+`, is defined using:

```
DEFINE OPERATION
    TYPE = CreateVlasovVariableOp
    DATA_NAME = VlasovFluidData
    VBOX_LO = (-6000.0,-0.5,-0.5)
    VBOX_HI = ( 6000.0, 0.5, 0.5)
    VBOX_NGHOST = [3 0 0]
    SPECIES_NAMES = fAr+
    SPECIES_COMPOSITION = Ar
```

Table 2.9: Summary of operations listed for the collisionless shock Vlasov example.

Stage	Operation	Description
INITIALIZE	`CreateVlasovVariableOp`	Create a Vlasov variable
	`LogicalVlasovFluidBoltzmannSetOp`	Sets initial conditions of a Vlasov variable using a Boltzmann distribution
MOVE	`LogicalVlasovCalcFluidVariablesOp`	Calculate field variables given a velocity distribution
	`LogicalBCVlasovExtrapolateOp`	Sets a velocity boundary conditions to extrapolation, i.e. simple non-reflecting
	`Vlasov1D1VSLOp`	Advects a Vlasov variable using the Semi-Lagrangian method
	`LogicalVlasovCalcDensityOp`	Calculates the density given a velocity distribution
	`LogicalFieldSetOp`	Set field values to constant
	`LogicalFieldAddOp`	Adds one field variable to another
	`LogicalFieldScalarMulOp`	Multiplies field by scalar constant
	`LogicalFieldVolumetricMulOp`	Multiplies or divides field data by cell volumes
	`LogicalBCXtrapOp`	Sets a physical boundary to extrapolation
	`LogicalBCConstantOp`	Sets a physical boundary to be a constant
	`LogicalPoissonBoltzmannStrip1DOp`	Solves for the electric filed assuming a Boltzmann electron
	`LogicalGradientCellCenterOp`	Calculates the gradient of a field vector
POSTOP	`LogicalVlasov2DWriterOp`	Exports a 2D phase-space plot
	`LogicalFieldWriteVTKROp`	Exports the filed data, e.g, density, velocity,...
	`VlasovMetricsOp`	Exports Vlasov metrics data e.g., mass and energy conservation

```
END OPERATION
```

The different fields are relatively self explanatory, but it should be noted that `VBOX_LO/HI` are in units of meters per second and the unused dimensions, V_y and V_z, need to have a length of one. Another important parameter is the `VBOX_NGHOST` which set the number of ghost cells in each velocity direction. If left unset, the default value of 3 ghost cells in the unused direction increases memory requirements by a factor of 49. The variable name is defined in `SPECIES_NAMES`, multiple species can be defined in this field. At a minimum the solver needs to know the mass and charge of each species. This is accomplished in one of two ways. The preferred method is to define the individual species that makes up each `SPECIES_NAMES`. This is defined in `SPECIES_COMPOSITION`. `SPECIES_COMPOSITION` will parse a chemical formula and calculate its mass and charge using an internal database of elements. Any values defined in `M` and `Z` will be ignored. The second method is to manually set the mass and charge via `M` in kg and `Z` respectively. For this method to work you must set the `SPECIES_COMPOSITION` to `None`. These two methods can be used together, but place holder will be needed in `M` and `Z` for the species defined via their `SPECIES_COMPOSITION`. For example the following code will use the internal database to calculate the mass and charge of `fAr+` and `fAr` but will set `FakeVar` using the values in `M` and `Z`. Note that place holders in `M` and `Z` are required.

```
SPECIES_NAMES = fAr+, FakeVar, fAr
SPECIES_COMPOSITION = Ar, None, Ar
M = -1.0, 1.0e-12, 23.3
Z = 1, -2, 0
```

This operator will not set the initial value of `fAR+` so we will need to call another operator to set the distribution. This is accomplished by:

```
DEFINE OPERATION
```

```
    TYPE = LogicalVlasovFluidBoltzmannSetOp
    PHASESPACE_TYPE = VlasovFluidData
    BOUND_LO = (-100.0e-3,-0.5,-0.5)
    BOUND_HI = ( 0.0e-3, 0.5, 0.5)
    PHASESPACE_NAME = fAr+
    TEMPERATURE_K = 2320.8 # 1.5ev
    NUMBER_DENSITY = 1.25e15
    INIT_ONLY = true
    NUMBER_OF_DIMENSIONS = 1
END OPERATION
```

This sets up the initial VDF with a Boltzmann distribution for the driver side. The target side is set using the same operator but with different `BOUND_LO/HI` values and density. The important parameters are `TEMPERATURE_K` and `NUMBER_DENSITY` which sets the temperature in Kelvin and the total number density per cell in m^{-3}. It should be noted that the molecular mass does not need to be defined because that information is stored within the variable `fAr+`. Another important parameter is `INIT_ONLY` which tells the operator that it should only run once at the beginning of the simulation. Otherwise the operation will overwrite the update with the initial conditions, though this could also be used as Dirichlet boundary condition at domain edges in other simulations.

Stage: Move

The next stage to run is the `MOVE` stage. This stage contains the advection of the fluid in phase space as well as the electric field solver and is broken up into four main steps: (1) X-advection (half Δt), (2) Electric field solver, (3) V_x-advection (full Δt), (4) X-advection (half Δt). This dimensionally split procedure was originally developed for the Vlasov equation by Cheng and Knorr and provides a second order integration in time [17]. The advection in phase space uses a Semi-Lagrangian method with WENO style interpolation and was developed by Qiu and Christlieb [18]. This method was chosen because it was found to be an accurate and efficient solver. The X-advection consists of two different operations, the first sets the ghost cells while the second advects the fluid in the X direction.

```
DEFINE OPERATION
    TYPE = LogicalBCVlasovExtrapolateOp
    NAME = PeriodicBCX1
    DATA_NAME = VlasovFluidData
    FIELD_NAME = fAr+
    DIRECTION = X
END OPERATION

DEFINE OPERATION
    TYPE = Vlasov1D1VSLOp
    NAME = Vlasov1D1VSL_X1
    DIRECTION = X
    VARIABLE_NAME = fAr+
    TIME_SCALE = 0.5
    VARIABLE_TYPE = VlasovFluidData
END OPERATION
```

It should be noted that `TIME_SCALE` is set to 0.5 which indicates that only a half time step should be taken.

The next set of operations are used to calculate the electric field using Boltzmann equilibrium electrons. Many of the variables are self explanatory but one parameter, `RUN_AT_INIT`, needs further explanation. `RUN_AT_INIT` signifies that the `apply` function of the operator should also be run during the initialization stage. Typically, during the initialization stage the operator will only parse the input file, create the required memory, and if necessary set the initial conditions. In most cases this is enough, but some variables, such as the electric filed, its value is not

explicitly known and a routine must be used to calculate it. Since the method used to calculate the initial electric field is the same used during the simulation it is more practical to define these operators once during the `MOVE` stage and set the `RUN_AT_INIT` to true.

```
DEFINE OPERATION
## Calculates several useful field variables from a velocity distribution
## including density, mean velocity and temperature. This routine may be used
## instead of LogicalVlasovCalcDensityOp
    TYPE = LogicalVlasovCalcFluidVariablesOp
    NAME = CalcFluidVariables
    PHASESPACE_TYPE = VlasovFluidData
    PHASESPACE_NAME = fAr+
    DENSITY_TYPE = FieldData
    DENSITY_NAME = rho_tmp
    MEAN_V_PREFIX = Vmean
    TEMPERATURE_NAME = Temperature
    MEAN_V_DIRECTIONS = [x, y, z]
    RUN_AT_INIT = true
END OPERATION

DEFINE OPERATION
## Calculates Density by integrating over velocity space
    TYPE = LogicalVlasovCalcDensityOp
    PHASESPACE_TYPE = VlasovFluidData
    PHASESPACE_NAME = fAr+
    DENSITY_TYPE = FieldData
    DENSITY_NAME = rhoI
    RUN_AT_INIT = true
END OPERATION

DEFINE OPERATION
## Sets electric potential (phi) to zero
    TYPE = LogicalFieldSetOp
    DATA_NAME = FieldData
    VALUE = 0.0
    FIELD_NAME = phi
    INIT_ONLY = false
    RUN_AT_INIT = true
END OPERATION

DEFINE OPERATION
## Add electron and ion density
   TYPE = LogicalFieldAddOp
   DATA_NAME = FieldData
   FIELD_SRCB_NAME = rhoE
   FIELD_SRCC_NAME = rhoI
   FIELD_DST_NAME = rhoE
   RUN_AT_INIT = true
END OPERATION

DEFINE OPERATION
## Multiples the ion density by the mass of an electron to convert
## from number density to mass density and set to rho_source
```

```
    TYPE = LogicalFieldScalarMulOp
    DATA_NAME = FieldData
    FIELD_SRC_NAME = rhoI
    FIELD_DST_NAME = rho_source
    SCALAR = 1.602189200e-19
    RUN_AT_INIT = true
END OPERATION

DEFINE OPERATION
## Finds mass of ions in each cell, i.e., no longer density
       TYPE = LogicalFieldVolumetricMulOp
       DATA_NAME = FieldData
       FIELD_NAME = rho_source
       RUN_AT_INIT = true
END OPERATION

DEFINE OPERATION
## Sets Electric field boundary conditions on the left hand side
       TYPE = LogicalBCXtrap
       NAME = Neumann-X-
       DATA_NAME = FieldData
       FIELD_NAME = phi
        BOUND_LO = (-120.0e-3,-1.0,-1.0)
        BOUND_HI = (-95.0e-3, 1.0, 1.0)
END OPERATION

DEFINE OPERATION
## Sets Electric field boundary conditions on the right hand side
       TYPE = LogicalBCConstantOp
       NAME = Electrode-X+
       value = 0.0 #
       DATA_NAME = FieldData
       FIELD_NAME = phi
       BOUND_LO = ( 95.0e-3,-1.0,-1.0)
       BOUND_HI = ( 120.0e-3,1.0, 1.0)
END OPERATION

DEFINE OPERATION
## Solves the Poison equation assuming Boltzmann electrons and 1D
    TYPE = LogicalPoissonBoltzmannStrip1DOp
    FIELD_NAME = phi
    SOURCE_NAME = rho_source
    NUMBER_DENSITY_REF_CGS = 1.0e9
    ELECTRON_TEMPERATURE_CGS = 3.0
    ELECTRON_DENSITY_NAME = Ne-
    NEUMANN_LEFT = TRUE
        SUBCYCLE = 1
# INIT_ONLY = TRUE
END OPERATION

DEFINE OPERATION
## Finds the electric filed from the gradient of the electric
```

```
## potential and multiplies by a constant
    TYPE = LogicalGradientCellCenterOp
    FIELD_DATA_NAME = FieldData
    FIELD_POTENTIAL_NAME = phi
    FIELD_GRADIENT_PREFIX = E
    FIELD_MULTIPLY_CONSTANT = -2.415365e6 #ec/(MW*amu) -2.415365e6
    FIELD_GRADIENT_DIRECTIONS = [x, y, z] ## Ex,Ey,Ez
    RUN_AT_INIT = true
    BOUNDARY_TYPE = EXTRAPOLATE
END OPERATION
```

The V_x-advection consists of two different operations, the first sets the ghost cells while the second advects the fluid in the V_x direction.

```
DEFINE OPERATION
    TYPE = LogicalBCVlasovExtrapolateOp
    NAME = PeriodicBCY
    DATA_NAME = VlasovFluidData
    FIELD_NAME = fAr+
    DIRECTION = VX
END OPERATION

DEFINE OPERATION
    TYPE = Vlasov1D1VSLOp
    NAME = Vlasov1D1VSL_Vx
    DIRECTION = VX
    VARIABLE_NAME = fAr+
    WAVE_SPEED_NAME = Ex
    TIME_SCALE = 1.0
    VARIABLE_TYPE = VlasovFluidData
END OPERATION
```

The final step of the `MOVE` stage is the second advection in the X-direction, which uses the same two operations used before.

```
DEFINE OPERATION
    TYPE = LogicalBCVlasovExtrapolateOp
    NAME = PeriodicBCX2
    DATA_NAME = VlasovFluidData
    FIELD_NAME = fAr+
    DIRECTION = X
END OPERATION

DEFINE OPERATION
    TYPE = Vlasov1D1VSLOp
    NAME = Vlasov1D1VSL_X2
    DIRECTION = X
    VARIABLE_NAME = fAr+
    TIME_SCALE = 0.5
    VARIABLE_TYPE = VlasovFluidData
END OPERATION
```

Stage: Postop

The third and final stage is the `POSTOP` stage. This stage is responsible for preparing and saving the results to various output files.

The first output file is a two-dimensional phase-space plot. The purpose of this operator is to take any two dimensions, X, Y, Z, V_x, V_y, or V_z and a coordinate in phase space and slice along those planes. This is controlled by the following variables, `SPACE_CORD`, `VELOCITY_CORD`, `X_PLOT_DIRECTION`, and `Y_PLOT_DIRECTION` which sets the spatial coordinate, phase space coordinate the coordinate to plot along the "X" axis and the phase space coordinate to plot along the "Y" axis respectively. This operator looks like:

```
DEFINE OPERATION
    INCLUDE_GHOST = false
    TYPE = LogicalVlasov2DWriterOp
    DATA_NAME = VlasovFluidData
    FILE_HEAD = shockdata/phase_
    FIELD_NAME = fAr+
    SKIP = 20
    SPACE_CORD = (0.0, 0.0, 0.0)
    VELOCITY_CORD = (-5.0, 0.0, 0.0)
    X_PLOT_DIRECTION = X
    Y_PLOT_DIRECTION = VX
    BINARY = false
    RUN_AT_INIT = true
END OPERATION
```

The next plotting operator is a bit of a hack and could be changed in future releases. The operator is designed to save the spatial data, e.g., density, electric field, and it is desirable to use the same operator regardless of the number of spatial dimensions. Unfortunately the VTK file format does not have a convenient mechanism to save one-dimensional data. To get around this an additional parameter `SAVE_AS_CSV` was added that saves the data in csv file format rather than the standard VTR format. The operator is set via:

```
DEFINE OPERATION
    TYPE = LogicalFieldWriteVTKROp
    DATA_NAME = FieldData
    FILE_HEAD = shockdata/field_data
    FIELD_NAME = rhoI, phi, Ex, rho_source, Vmeanx, Vmeany, Vmeanz, Temperature, Ne-
    SKIP = 10
    DIMENSIONS = 2
    nFIELD_NAME = 4
    RUN_AT_INIT = true
    SAVE_AS_CSV = true
END OPERATION
```

The final operator in this stage is used to save various metrics including the density, energy, entropy and electric filed norms over time. The operator is set up via:

```
DEFINE OPERATION
    TYPE = VlasovMetricsOp
    PHASESPACE_TYPE = VlasovFluidData
    SPACE_TYPE = FieldData
    PHASESPACE_NAME = fAr+
    DENSITY_NAME = rhoI
    E_FIELD_PREFIX = E
    E_FIELD_DIRECTIONS = [x, y, z]
```

```
    FILE_NAME = shockdata/norms.csv
    SKIP = 1
    RUN_AT_INIT = true
END OPERATION
```

CHAPTER 3

TURF INFRASTRUCTURE RELEASE 2017A

Samuel J. Araki, Robert Martin, and Kari A. Kawashima

Contents

3.1 Capabilities

Samuel J. Araki

Table 3.1 summarizes the tutorials added for TURF-IR v2017a, and Table 3.2 provides the list of newly added operations. The major difference of v2017a from 2016 is the introduction of `MSPDist` class object that is upgraded from `SPDist`. Unlike `SPDist`, `MSPDist` holds species ID, enabling multi-species simulation with a single `MSPDist` class object. This capability is particularly important for a spacecraft integration simulation that involves several different gas species from an electric propulsion (EP) device and spacecraft walls caused by high-energy heavy particles impact. In addition to species ID, `MSPDist` now holds particle tag, particle position from one time-step earlier, and species mass and charge. The particle tag is useful when displaying particle trajectories in ParaView (see Section 3.7). By storing the particle position at the previous time-step, intersection of particle and geometry surface can be more easily and rigorously determined. Many of the `SPDist` operations were rewritten to be compatible with `MSPDist`. TURF-IR v2017a contains the same tutorials as v2016 such as heatbath, grounded box, and one-dimensional normal shock except that the `SPDist` operators are replaced with `MSPDist` operators.

TURF-IR v2017a also includes several `MSPDist` class operations that are useful for EP plume simulations. These include charge deposition (`MSPDistChargeDepositionOp`), elastic collision including momentum- and charge-exchange collisions (`MSPDistMCCElasticFitOp`), output of the particle distribution (`MSPDistWriteVTKOp`), numerical probes (`MSPDistProbeFixedOp` and `MSPDistProbeStageSphericalOp`), and so on. With these newly added operators, TURF-IR can be used to perform a simplified EP plume simulation. However, TURF-IR lacks an important physics modules that enables the real-world plume simulations. In particular, TURF-IR does not include an interface to extract particles from a high fidelity thruster model such as HPHall. The alternative approach is to inject charged species according to the pre-defined current distribution using `MSPDistSourceRPAOp`, which is demonstrated in the EP plume tutorial.

Table 3.1: Tutorials provided for TURF-IR v2017a.

Folder	Description of Problem	Type of Solver	Section
Heatbath_MS	Free molecular flow in a specular box	Particle	3.2
GroundedBox_MS	Plasma in a grounded box	PIC	3.3
1DShock_MS	One-dimensional normal shock wave	DSMC	3.4
Plume	Electric propulsion plume simulation	PIC	3.5-3.7

Table 3.2: Summary of operations included in TURF-IR v2017a.

Module	Operation	Description
DSMC	`MSPDistDSMCOp`	DSMC collision calculation
Field	`FieldArithmeticOp`	Perform arithmetic operation to two fields
Field	`FieldBlendTimeOp`	Blend two fields over time
Field	`FieldSetOp`	Set unstructure or structure field data
Field	`LogicalFieldPatchOp`	Write to output files for 3D plots in .vts format
Field	`LogicalPotentialBoltzmannOp`	Find potential from Boltzmann relations
Field	`LogicalPotentialSetInsideGeometryOp`	Integer flag for regions in/out body to select potential solver
Field	`UFieldMSInitOp`	Create unstructured field data
Geometry	`LogicalMeshGlobalRecolorOp`	Complete sugarcubing for multi-domain simulation
Geometry	`LogicalMeshSurfaceSugarcubeOp`	Create sugarcube surface mesh/structured mesh intersection
Geometry	`MSPDistSugarcubeSurfIntersectionOp`	Determine if particles intersect with surface elements
Geometry	`SurfaceComponentSetOp`	Add component list to surface mesh object
Geometry	`SurfaceMeshMergerOp`	Merge two or more surfaces meshes
Geometry	`UMeshImporterOp`	Read unstructured volume or surface mesh file
Particle	`MSPDistBCSpecOp`	Specular boundary condition for particles
Particle	`MSPDistBoxICOp`	Initialize particles uniformly in physical box
Particle	`MSPDistCellIDOp`	Find cell ID associated with particle location
Particle	`MSPDistChargeDepositionOp`	Accumulate charge to cells
Particle	`MSPDistCombineOp`	Combine particles from different distributions
Particle	`MSPDistConstantICOp`	Create uniform constant number of particles/cell within a box
Particle	`MSPDistCopyOp`	Copy srcdist to dstdist, and set srcdist particle count to zero
Particle	`MSPDistESPushOp`	Electrostatic particle push using node electric field
Particle	`MSPDistInitOp`	Create new particle distribution of MSPDist class
Particle	`MSPDistMCCElasticFitOp`	Apply MCC elastic collision
Particle	`MSPDistMoveOp`	Advance particles
Particle	`MSPDistParticleCountOp`	Obtain global count of particles
Particle	`MSPDistReadVTKOp`	Read MSPDist from MSPDistWriteVTKOp VTK output
Particle	`MSPDistRemovePartOp`	Remove particles in MSPDist with w=0 or cellID>mesh->Ncells
Particle	`MSPDistSampleOp`	Sample particles to obtain field data.
Particle	`MSPDistSortOp`	Sort MSPDist particles for cellID
Particle	`MSPDistSplitOp`	Split particle distributions
Particle	`SampleFieldReadVTKOp`	Read sample field data from SampleFieldWriteVTKOp output
Particle	`SampleFieldWriteVTKOp`	Write MSPDistSampleOp field data to VTK file format
Plotting	`LogicalFieldWriteVTK2DOp`	Plot 2D slice of 3D data
Plotting	`MSPDistWriteVTKOp`	Output complete MSPDist data for restart capability
Plotting	`UFieldWriteVTKOp`	General unstructured field writer
Probe	`MSPDistProbeFixedOp`	Faraday/RPA numerical probes attached to a surface
Probe	`MSPDistProbeStageSphericalOp`	Faraday/RPA numerical probes on a virtual spherical stage
Probe	`ProbeFixedWriteOp`	Write fixed probe data
Probe	`ProbeStageWriteOp`	Write spherical stage probe data
SourceModel	`MSPDistNormalMaxwellianOp`	Inject particle Maxwellian from surface mesh or virtual surface
SourceModel	`MSPDistNormalMaxwellianStreamOp`	Maxwellian stream source from triangulated surfaces
SourceModel	`MSPDistSourceRPAOp`	Inject particle according to the current density profile
Surface	`MSPDistSurfaceInteractionOp`	Perform reflection and sputtering particle-surface interactions
Surface	`UFieldMSComputeOp`	Compute surface field
Utility	`GSOPatchOp`	Inter-domain particle patch

3.2 Heatbath with Multi-Species Particle Data

Samuel J. Araki

3.2.1 Introduction

This tutorial is the simplest among all the other examples in the Thermophysics Universal Research Framework (TURF) version 2017a and should be the first one to be covered. This tutorial replaces the heatbath example for TURF version 2016 provided in `tutorial-TURF/TURF-IR_2016/Heatbath` and only uses operators that utilize MSPDist class object[1]. In the directory `tutorial-TURF/TURF-IR_2017a /Heatbath_MS/Particle`, there are seven files ending with `.list` extension. These files serve as script files on TURF. The script files starting with `world` in their file names, `world.*.list`, include the global parameters that define the entire simulation such as the simulation size, the number of stages, etc. These files point to one or more `operation.*.list` files that contain individual TURF operations within each stage.

In this example, the complexity of the problem is incremented in stages to help readers understand how to build a simulation in TURF. In particular, this example demonstrates how to create, move, remove, output particles, and apply boundary condition to them. With all the operators turned on, it will simulate a free molecular flow in a specular box; we will call this the "heatbath" example. Initially, thermal particles are uniformly distributed within a box smaller than the simulation domain. These particles fill the empty space and eventually be distributed uniformly within the whole simulation domain. This tutorial also demonstrates how to extend the simulation to multi-domain simulation.

3.2.2 Setting up a simulation

When a simulation starts, TURF always looks for a file named as `world.list`, while this tutorial has two world scripts with slightly different names, `world.heatbath.list` and `world.heatbathx2.list`. Without `world.list`, TURF terminates immediately with an error message. In this situation, a symbolic link between `world.list` and one of two file can be made, such that `world.list` is routed to the proper world file. A symbolic link in the Linux terminal can be created by

```
tutorial-TURF/TURF-IR_2017a/Heatbath_MS/Particle> ln -s world.*.list world.list
```

where * can be "heatbath" or "heatbathx2" for this example. When `world.list` already exists in the directory, `-sf` option can be used to force replacing the existing link.

Once the proper world script is linked to `world.list`, TURF can be run from the directory. For a single domain/single MPI process simulation, simply typing the binary name with its absolute or relative path from the directory will start the simulation. For multiple MPI processes simulation, `mpiexec` command can be used before the binary. Assuming that the binary, `TURF-o`, is created in bin directory under TURF folder, the command to run TURF is,

```
tutorial-TURF/TURF-IR_2017a/Heatbath_MS/Particle> mpiexec -n x ./../../../../bin/TURF-o
```

where `x` is the number of cores to run.

The world script contains inputs for a "World" object such as global information about the simulation as well as a list of sub-domains in which the computational domain has been partitioned. Each MPI process possesses one World object, and each World object possesses one or more sub-domains. The script file, `world.heatbath.list`, is shown below. In order to run a case, an input to `OP_FILE` should be changed to coincide with one of the operation file listed in Table 3.3. In this script, simulation time-step (`START_DT`), end time (`END_TIME`), origin of the coordinate system (`ORIGIN`), and volume mesh cell size (`DELTA`) are defined. TURF is written to assume all units are in MKS. With this in mind, the number of iterations is 250 for all the examples in this tutorial. In TURF, the number of grid cells is determined by finding how many `DELTA`s fit in the user-defined domain bounds (`BOUND_HI-BOUND_LO`), and the domain bounds are redefined such that the coordinates of the lowest corner of the domain box are placed at `ORIGIN`. Therefore, the domain box may not necessarily be bounded by the user-defined `BOUND_HI-BOUND_LO`. In this example, the grid contains 32 interior cells in each directions. On top of this, layers of ghost cells are added around

[1] MSPDist class object is similar to SPDist but holds additional information such as species ID, particle index, and position from last time-step

the interior mesh[2]. Here, BOUND_HI is set to slightly larger numbers than intended; this ensures that the domain size is not reduced by one cell due to the rounding down of the number of cells. Only helium gas (He@g) is used in this simulation, and the field parameters to be computed are N, NC, and NP as defined in Table 3.4. This example uses two stages (INITIALIZE and MOVE), and MPI communication and synchronization is performed between the stages. Note that the names for fields and stages are only labels and do not refer to any existing information in the code. On the other hand, the material name must correspond to the name or composition defined in the material database file provided in src-TURF/src/Materials/database/materials.list. However, it is important to reference the same name if they are used elsewhere in the code . Finally, there are more options for the world script such as PRINT_STATS_SKIP[3], PRINT_PROFILE_INFO[4], and SAVE_PROFILE_INFO[5].

```
DEFINE WORLD
      NAME = Heatbath-Example
      OP_FILE = operations.list # Replace the file name with the one to run
      COORDINATES = cartesian
      ORIGIN = (0.0,0.0,0.0)
      DELTA = (100.0e-6,100.0e-6,100.0e-6)
      END_TIME = 250.1e-9
      START_DT = 1.0e-9
      FIELDS = [ NC NP N ]
      MATERIALS = [ He@g ]
      STAGES = [INITIALIZE, MOVE]
END WORLD

###############################################################################
## Domain Geometry
###############################################################################
DEFINE DOMAIN DOM000
      BOUND_LO = (0.0,0.0,0.0)
      BOUND_HI = (3.2001e-3,3.2001e-3,3.2001e-3)
END DOMAIN
```

3.2.3 Operations

Table 3.5 lists all the operations used in this tutorial for the complete heatbath example. In this tutorial, a combination of operations is added in stages in the order listed in Table 3.3. Text file comparison utilities such as Meld[6] are very useful in finding the new operations added between the examples. Outputs from TURF are typically in Visualization ToolKit (VTK) format. Therefore, it is recommended that users have access to scientific visualization software such as ParaView [19] and VisIt [20]. Some of the ParaView commands to visualize TURF output files are introduced in Section 3.7.

(A) Go to Next Stage

The first example, operations.nextstage.list, demonstrates a case with minimal number of operations. In this example, TURF constructs the World object but performs nothing else until the simulation reaches the final iteration. Within each stage, an operation that tells TURF to advance to next stage is required. This example uses an operation NextStageOp, and TURF advances to next stage when all the operations defined in the stage are executed. If all the operations within the stage are to be repeated until a certain criterion is met, CriteriaStageOp can be used in place of NextStageOp. It is important to note that, without one of the two operations in each

[2] Three ghost layers are added by default, but this can be changed by defining GHOST_CELLS in the DEFINE DOMAIN block.
[3] PRINT_STATS_SKIP: Prints timing info every x number of iteration, where x is defined by PRINT_STATS_SKIP
[4] PRINT_PROFILE_INFO: Prints profiling information at the end of simulation if defined as TRUE
[5] SAVE_PROFILE_INFO: Saves profiling information if defined as TRUE
[6] http://meldmerge.org

Table 3.3: Operation list files and added operations from the last.

Operation List File	Description of the Simulation	Example
operations.nextstage.list	Time advances but code does nothing	A
operations.addparticles.list	Add particles in a box	B
operations.writeoutput.list	Write vtk field output files periodically	C
operations.push-untrimmed.list	Push particles forward in Time	D
operations.push.list	Trim escaped particles from distribution	E
operations.heatbath.list	Add Reflecting Boundary Conditions at Domain Bound	F
operations.heatbathx2.list	Particle patching between sub-domains	G

Table 3.4: Volume mesh fields[a] computed in this heatbath example.

Field Name	Description
NC	Number of simulation particles.
NP	Number of physical particles.
N	Density.

[a] For multi-species simulations, field data for each species can be obtained by adding species name followed by a underscore. For example, if there are two species A@g and B@g, partial densities for species A@g and B@g and total density are named as N_A@g, N_B@g, and N, respectively.

Table 3.5: List of operations used in this tutorial.

Stage	Operation	Description
INITIALIZE	`MSPDistInitOp`	Create MSPDist objects (`P-DST`, `P-GST`, and `P-EXC`)
	`MSPDistBoxICOp`	Fill a box with particles and add to `P-DST`
	`MSPDistCombineOp`	Combine `P-DST` and `P-EXC` to complete patching
	`MSPDistCellIDOp`	Find cell ID associated with particle location. Particles outside the sub-domain are marked, but there should not be any here in this example.
	`MSPDistRemovePartOp`	Remove particles outside of entire simulation domain
	`NextStageOp`	Go to next stage
MOVE	`MSPDistWriteVTKOp`	Write `P-DST` particle data to a VTK file
	`MSPDistSampleOp`	Sample `P-DST` particles and compute volume mesh field data
	`LogicalFieldWriteVTKOp`	Output 3D field data to a VTK file
	`MSPDistMoveOp`	Advancement of `P-DST` particles
	`MSPDistBCSpecOp`	Apply specular boundary condition
	`MSPDistCellIDOp`	Find cell ID associated with particle location. Particles outside the sub-domain are marked.
	`MSPDistSplitOp`	Split the `P-DST` particles outside of sub-domain to `P-EXC`
	`GSOPatchOp`	Pass `P-EXC` across sub-domains via MPI communication
	`MSPDistParticleCountOp`	Output particle count
	`NextStageOp`	Go to next stage

stage, TURF never gets out of the stage, causing the program to simply hang. It should also be noted that the stage names INITIALIZE and MOVE are only names. Despite being named "INITIALIZE," this stage is called every iteration.

```
################################################################################
DEFINE STAGE INITIALIZE
################################
## Proceed to the Next Stage ##
################################
DEFINE OPERATION
      TYPE = NextStageOp
END OPERATION

END STAGE INITIALIZE

################################################################################
DEFINE STAGE MOVE
################################
## Proceed to the Next Stage ##
################################
DEFINE OPERATION
      TYPE = NextStageOp
END OPERATION

END STAGE MOVE

################################################################################
```

(B) Add Particles

The second example, operations.addparticles.list, creates a MSPDist object that holds particle information (MSPDistInitOp) and adds particles uniformly within a user-specified box (MSPDistBoxICOp). When calling MSPDistInitOp, its name (MSPDIST_DATA_NAME), the buffer size (MAX_NP), and species held by the MSPDist object (SPECIES_NAMES) are required. It is important to ensure that the number of particles does not exceed the buffer size, otherwise TURF crashes with an error message in the course of the simulation. The second operation, MSPDistBoxICOp, defines the bounds of a box (BOUND_LO and BOUND_HI) and distributes particles with a given temperature (TEMPERATURE) and drift velocity (VEL) inside it such that it fulfills the number density (NUMBER_DENSITY). When the INIT_ONLY option is on, particles are created only during the first iteration. The MSPDistBoxICOp, by default, truncates the user-defined box such that it fits within the simulation domain, which can be turned off by enabling the BOX_OUTSIDE_DOMAIN option.

```
################################
## Create Particle Distibution ##
################################
DEFINE OPERATION
      TYPE = MSPDistInitOp
      MSPDIST_DATA_NAME = P-DST
      MAX_NP = 1280000
      SPECIES_NAMES = He@g
END OPERATION

################################
## Fill a Box with Particles ##
```

```
##############################
DEFINE OPERATION
      TYPE = MSPDistBoxICOp
      MSPDIST_DATA_NAME = P-DST
      SPECIES = He@g
      BOUND_LO = (0.10e-3,0.10e-3,0.10e-3)
      BOUND_HI = (2.32e-3,2.32e-3,2.32e-3)
      TEMPERATURE = 11604.5059
      NUMBER_DENSITY = 1.0e14
      REAL_TO_COMPUTATIONAL = 10.0
      VEL = (0.0,0.0,0.0)
      INIT_ONLY = TRUE
      BOX_OUTSIDE_DOMAIN = FALSE
END OPERATION
```

(C) Output Field Data and Particles

The third example, `operations.writeoutput.list`, samples particles to compute volume mesh field data (`MSPDistSampleOp`) and outputs field data (`LogicalFieldWriteVTKOp`), particle information (`MSPDistWriteVTKOp`), and the number of particles (`MSPDistParticleCountOp`). No other operation is applied to particles such that particles remain to be exactly as the initial state. `MSPDistSampleOp` samples mass, momentum, and kinetic energy from a particle distribution and computes volume mesh field data from these sampled quantities. The quantity to be sampled depends on `FIELD_NAMES` defined in the input. The frequencies of sampling and field calculation are defined by `STEPS_PER_SAMPLING` and `STEPS_PER_FIELD_CALCULATION`. `STEPS_PER_SAMPLING` is typically set to a value greater than one to remove the correlation of the flow field caused by numerics (e.g. particle injection). `STEPS_PER_FIELD_CALCULATION` should match with the output frequency defined by `SKIP` in `LogicalFieldWriteVTKOp`. After the time defined by `MIX_START_TIME`, sampled quantities are blended between iterations in order to improve the statistics. Therefore, this operation allows a smooth field distribution at the end of simulation without using a very large number of simulation particles. Here, `MIX_START_TIME`, should be set to the time when the simulation becomes steady-state.

In TURF, layers of ghost cells are added to the domain, but the field data within this region are not computed by `MSPDistSampleOp`. Although `LogicalFieldWriteVTKOp` outputs the data within the ghost region as well as the interior region, it can be turned off by setting `PLOT_GHOST` to `FALSE`. The output of the ghost region can be useful for debugging purpose. `MSPDistWriteVTKOp` is used to output particle information such as particle position, velocity, index, and species. The VTK files are output every some number of iteration defined by `SKIP`, and the corresponding iteration number is added to the file name. The same file can be overwritten by turning the `OVERWRITE` option on such that the disk space can be saved. Any of the output from this operation can be used to restart the simulation by `MSPDistReadVTKOp`. The particle output can be written in ASCII or binary format by specifying one for `FORMAT`. If some fraction of the particles are to be output, `PARTICLE_SKIP` can be set to a value larger than one. Figure 3.1(a) shows the initial particle distribution. The particle distribution remains the same at the final time in this example.

```
##############################
## Sum to Fields for Output ##
##############################
DEFINE OPERATION
      TYPE = MSPDistSampleOp
      FIELD_DATA_NAME = FieldData
      MSPDIST_DATA_NAME = P-DST
      FIELD_NAMES = NC NP N
      MIX_START_TIME = 1.0e-2
      STEPS_PER_SAMPLING = 1
      STEPS_PER_FIELD_CALCULATION = 5
```

```
END OPERATION

######################
## Write Field Data ##
######################
DEFINE OPERATION
      TYPE = LogicalFieldWriteVTKOp
      FIELD_DATA_NAME = FieldData
      FILE_HEAD = plot3D/plt_
      FIELD_NAMES = NC NP N
      PLOT_GHOST = FALSE
      SKIP = 5
END OPERATION

######################
## Output Particles ##
######################
DEFINE OPERATION
      TYPE = MSPDistWriteVTKOp
      MSPDIST_DATA_NAME = P-DST
      SPECIES = ALL
      FILE_HEAD = particle/MSPDist_
      SKIP = 5
      PARTICLE_SKIP = 1
      OVERWRITE = FALSE
END OPERATION

############################
## Output particle count ##
############################
DEFINE OPERATION
      TYPE = MSPDistParticleCountOp
      MSPDIST_DATA_NAME = P-DST
      LOG_FILE = ParticleCount.csv
      VERBOSE = FALSE
      GLOBAL = TRUE
END OPERATION
```

(D) Move Particles

The fourth example, `operations.move.list`, advances particles to the time corresponding to the next iteration (`MSPDistMoveOp`)[7]. `MSPDistMoveOp` is the simplest particle time integrator that uses the explicit first-order Euler method and does not take into account the external force such as electric and magnetic fields. If charged particles are to be advanced in an electric field, `MSPDistESPushOp` should be used instead. Other methods for pushing particles can be implemented in TURF easily. Figure 3.1(b) shows the particle distribution at the final time. In this example, the particle distribution thermally expands over time after initializing the particles inside a cube, as there is not an operation to remove or force particles to stay within the domain.

```
################################
## Advance Particle Positions ##
```

[7] The time to advance a particle is computed by $t_i + \Delta t - t_p$ where t_i is the current iteration time, Δt is the time-step between iterations $i+1$ and i, and t_p is the particle time at the current position.

```
################################
DEFINE OPERATION
      TYPE = MSPDistMoveOp
      MSPDIST_DATA_NAME = P-DST
      SPECIES_NAMES = He@g
END OPERATION
```

(E) Remove Particles

The fifth example, `operations.remove.list`, determines the volume mesh cell that particles reside (`MSPDistCellIDOp`) and removes particles from the MSPDist object based on the cell ID (`MSPDistRemovePartOp`). Without these operations, particles can fly out of the simulation domain and are never deleted from the particle list. `MSPDistCellIDOp` marks particles outside the interior domain by assigning a large integer, typically the maximum cell index plus one. Then, `MSPDistRemovePartOp` gets rid of these particles from the MSPDist object if `REMOVE_CELLID_GT_MAX` option is turned on. The particles can also be removed based on their weights by turning `REMOVE_ZERO_WEIGHT` option on. In order to remove particles, the particle list is first copied from `P-DST` to `P-GST`, and then only the ones to be kept are copied back to `P-DST`. In this way, the operation can be executed in parallel with OpenMP or CUDA. The deletion of particles can also be done with `MSPDistSortOp`, but this operation sorts particles based on their cell IDs in addition to removing particles[8]. Figure 3.1(c) shows the particle distribution at the final time.

```
#####################################################
## Setting cellID==Max for the ones out of domain ##
#####################################################
DEFINE OPERATION
      TYPE = MSPDistCellIDOp
      MSPDIST_DATA_NAME = P-DST
END OPERATION

#############################################
## Remove particles if outside the domain ##
#############################################
DEFINE OPERATION
      TYPE = MSPDistRemovePartOp
      MSPDIST_SRC_NAME = P-DST
      MSPDIST_DST_NAME = P-GST
      REMOVE_ZERO_WEIGHT = FALSE
      REMOVE_CELLID_GT_MAX = TRUE
      DISCARD_REMOVED_PARTICLE = TRUE
      VERBOSE = FALSE
END OPERATION
```

(F) Apply Boundary Condition

The previous example demonstrated how to remove particles leaving the simulation domain. However, the two operations, `MSPDistCellIDOp` and `MSPDistRemovePartOp`, are not necessarily required for a heatbath simulation if particle boundary condition is applied properly. The sixth example, `operations.heatbath.list`, demonstrates the use of the simplest particle boundary condition implemented in TURF (`MSPDistBCSpecOp`). This operation reflects particles specularly from one of the six surfaces of a box defined by `BOUND_LO` and `BOUND_HI`. The surface

[8] The Direct Simulation Monte Carlo collision calculation can be more efficient when particles are sorted in advance. For this case, `MSPDistSortOp` is used in place of `MSPDistRemovePartOp`

to reflect particles is defined by `DIRECTION`, which can be specified as `xm`, `xp`, `ym`, `yp`, `zm`, and `zp`, and the 'm' and 'p' in these directions refer to "minus" (from negative to positive direction) and "plus" (vise versa), respectively. In the heatbath example, particles are reflected off six different surfaces such that `MSPDistBCSpecOp` is required to be defined six times, each with different inputs for `DIRECTION`, `BOUND_LO`, and `BOUND_HI`. It is important to note that particles may escape the domain if they never land within these boxes defined for `MSPDIstBCSpecOp`. In order to prevent this situation, the box has to be sufficiently large and/or the particle time-step has to be sufficiently small. Furthermore, the box is placed such that it slightly overlaps the simulation domain. This is to prevent particles landing exactly on the domain edge to be lost. The specular reflection can also be applied by `MSPDistSurfaceInteractionOp`, but this operation requires a surface mesh. Figure 3.1(d) shows the particle distribution at the final time. By introducing the boundary condition, particles are distributed nearly uniformly at the end of simulation.

```
#########################
## Boundary Conditions ##
#########################
# Specular boundary for +x to make the problem 1D
DEFINE OPERATION
       TYPE = MSPDistBCSpecOp
       MSPDIST_DATA_NAME = P-DST
       SPECIES_NAMES = He@g
       DIRECTION = xm
       BOUND_LO = (-10.0e-4,-10.0e-4,-10.0e-4)
       BOUND_HI = (0.00001e-4, 42.0e-4, 42.0e-4)
END OPERATION
# Specular boundary for -x to make the problem 1D
DEFINE OPERATION
       TYPE = MSPDistBCSpecOp
       MSPDIST_DATA_NAME = P-DST
       SPECIES_NAMES = He@g
       DIRECTION = xp
       BOUND_LO = (31.9999e-4,-10.0e-4,-10.0e-4)
       BOUND_HI = (42.0e-4, 42.0e-4, 42.0e-4)
END OPERATION
```

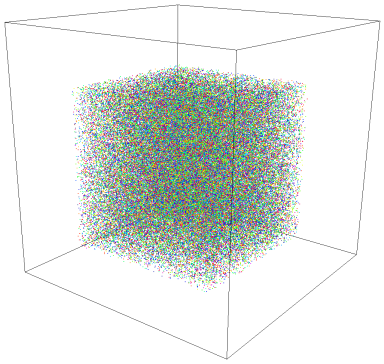
(G) Particle Patching with MPI Communication

The last example, `operations.heatbathx2.list`, demonstrates how to extend the single domain heatbath example to two sub-domains. In order to run this example, `world.heatbathx2.list` should be linked to `world.list`. In `world.heatbathx2.list`, the simulation domain is set up as follows.

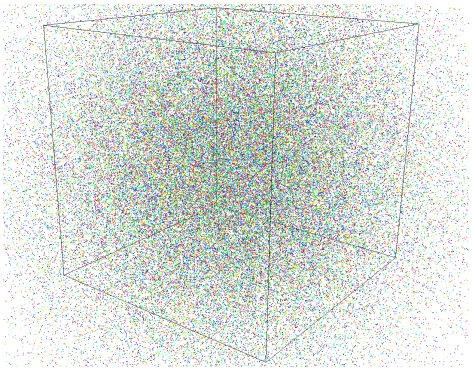
```
DEFINE DOMAIN DOM
       BOUND_LO = (0.0,0.0,0.0)
       BOUND_HI = (3.2001e-3,3.2001e-3,3.2001e-3)
       SUB_DOMAINS = (2,1,1)
END DOMAIN
```

This simply divides the domain evenly by the integer number input for `SUB_DOMAINS` in each direction. The same domain decomposition can be achieved by manually defining each domain.

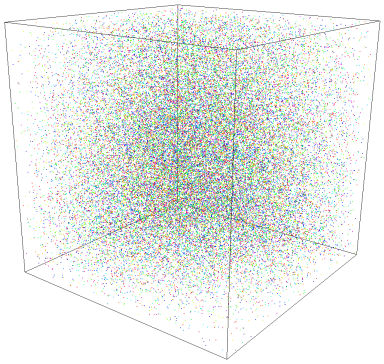
```
DEFINE DOMAIN DOM000
       BOUND_LO = (0.0,0.0,0.0)
       BOUND_HI = (1.6e-3,3.2e-3,3.2e-3)
```



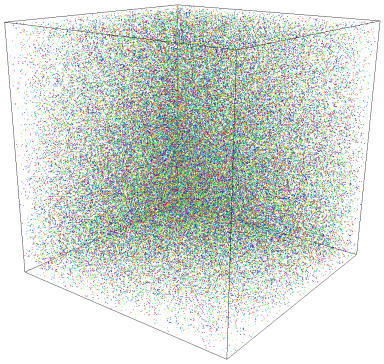
(a) Initial particle distribution (C).



(b) Particles at the final iteration (D).



(c) Particles at the final iteration (E).



(d) Particles at the final iteration (F).

Fig. 3.1: Initial and final particle positions from different examples. These outputs are from ParaView.

```
END DOMAIN
DEFINE DOMAIN DOM001
       BOUND_LO = (1.6e-3,0.0,0.0)
       BOUND_HI = (3.2001e-3,3.2001e-3,3.2001e-3)
END DOMAIN
```

When running with multiple sub-domains, additional operations are required to properly exchange particle information between the sub-domains. In TURF, the sub-domains do not overlap with each other. However, the ghost cell layers around a sub-domain can overlap with other sub-domains or ghost layers. TURF passes the information stored in the ghost layers to a sub-domain through MPI communications. Since MPI communication is performed between stages, particle patching is performed across at least two stages. In performing particle patching, a new particle distribution (`P-EXC`) is required. The buffer size for `P-EXC` should be sufficient but small for efficient communication between processes. If a particle moves between the two domains, `MSPDistSplitOp` moves it from `P-DST` and temporarily places it into the `P-EXC` distribution until the beginning of the next iteration. The particles in `P-EXC` can be within ghost cell region which overlaps with a sub-domain, and for those particles, `GSOPatchOp` tells TURF to pass them to a process that owns the sub-domain. The actual communication is done at the end of stage where `GSOPatchOp` is defined. Now, `P-EXC` contains particles that reside in the actual sub-domain (not in a ghost region or outside the whole domain). Finally, these particles are combined with the main particle distribution, `P-DST`, through `MSPDistCombineOp`. Finally, `MSPDistRemovePartOp` is called afterward to get rid of particles that escaped from the entire domain.

```
#####################################################
```

```
## Setting cellID==Max for the ones out of domain ##
#####################################################
DEFINE OPERATION
       TYPE = MSPDistCellIDOp
       MSPDIST_DATA_NAME = P-DST
END OPERATION

########################################
## Move cellID==Max particles to P-EXC ##
########################################
DEFINE OPERATION
       TYPE = MSPDistSplitOp
       MSPDIST_SRC_NAME = P-DST
       MSPDIST_DST_NAME = P-EXC
END OPERATION

####################
## Patch MSPDist ##
####################
DEFINE OPERATION
       TYPE = GSOPatchOp
       SRC_NAME = P-EXC
       DST_NAME = P-EXC
END OPERATION
```

```
###################################
## Combine MSPDist after patching ##
###################################
DEFINE OPERATION
       TYPE = MSPDistCombineOp
       MSPDIST_SRC_NAME = P-EXC
       MSPDIST_DST_NAME = P-DST
       VERBOSE = FALSE
END OPERATION
```

3.2.4 Useful Tools

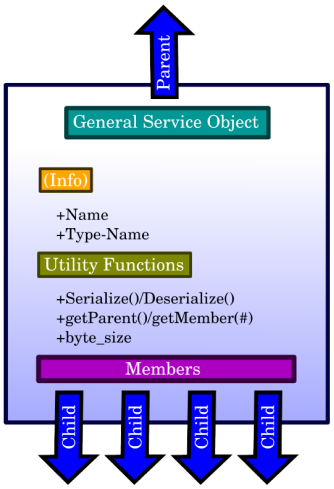


Fig. 3.2: Schematic of GSObject.

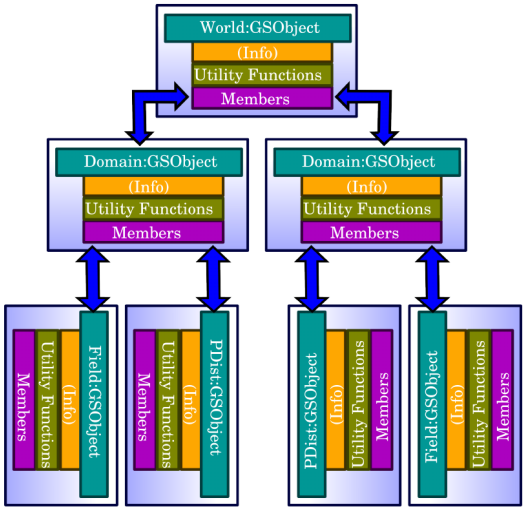


Fig. 3.3: Tree-hierarchy structure of objects with GSObject.

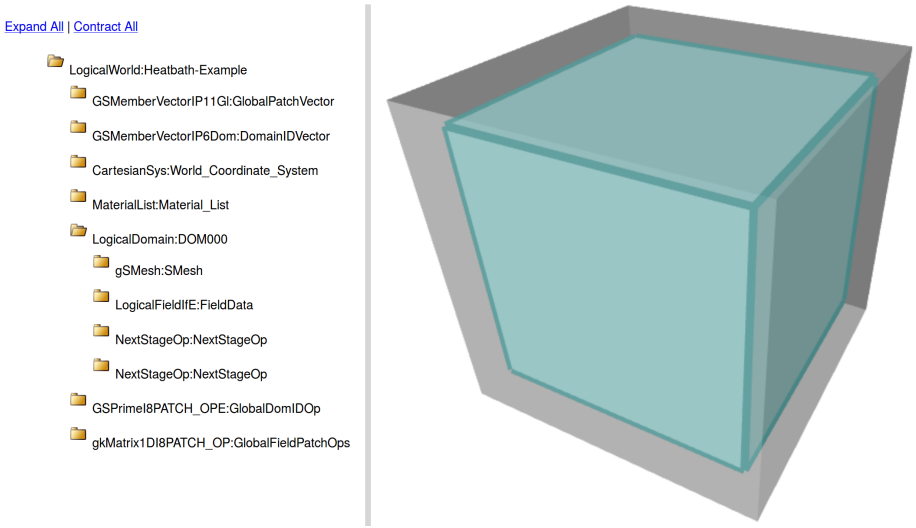


Fig. 3.4: Screenshot of World-Rank.html.

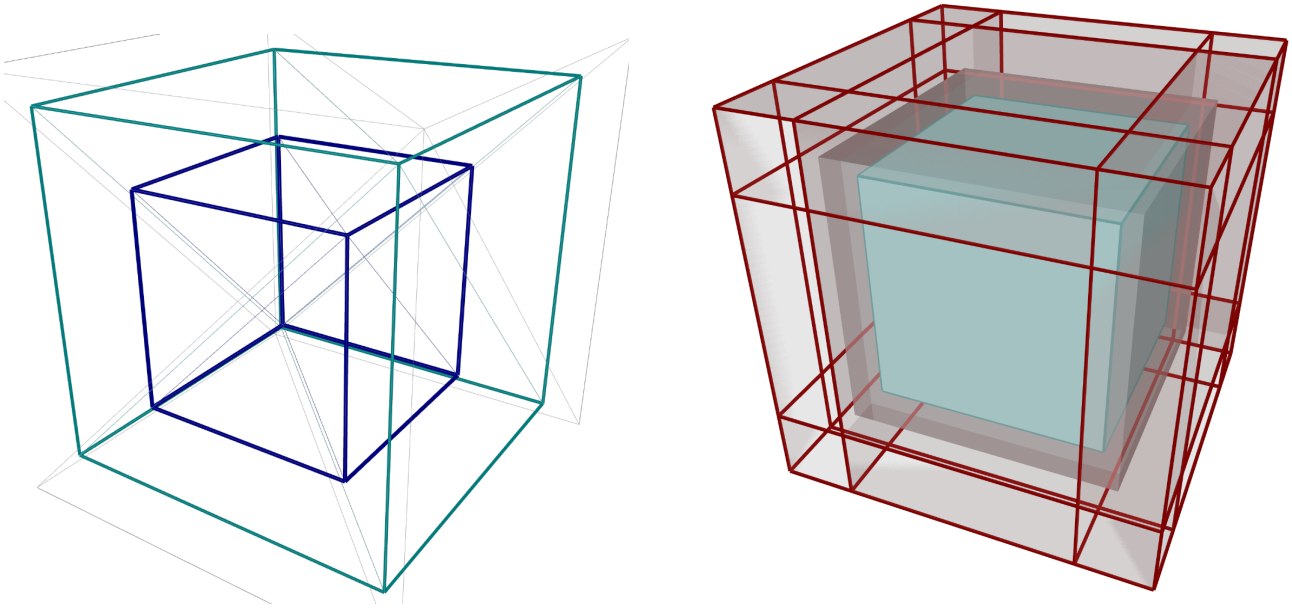
Visualizing TURF Tree-Hierarchical Structure

In TURF, a CPU core can perform calculations for multiple sub-domains. Every CPU has its own "World" object that contains global information about the simulation as well as the list of sub-domains. Then, every sub-domain contains a list of objects (i.e. particle distributions, fields, and operations) in addition to its size and index. This tree-hierarchy structure is a key element of the framework, which is attained using the General Service Object (GSObject) or a branching double-linked list with member functions as shown in Fig. 3.2. A GSObject includes "name" and "type-name" information to enable high-level search and access routine. In TURF, most of class objects are inherited from GSObject to form the tree-hierarchy structure with the World object as a common ancestor (see

Fig. 3.3).

When TURF is run, a html file named `World-Rank.html` is automatically generated, in which, the user can view the object hierarchy of the example (See Fig. 3.4). At the base of the tree is the logical world, which was named Heatbath-Example. The branches include `GSObject` named `GSMemberVector` (which have the functionality of a vector and can be used by the GPU), a material database, a logical domain, and the coordinate system defined by the `world.list` file. It is possible to expand the hierarchy to investigate any underlying databases or arrays which are automatically generated.

Another useful feature is the visualization of the simulation environment as shown in Fig. 3.4. The green cubic box represents the simulation domain, and the surrounding gray region is a layer of three ghost cells which are automatically generated when the domain is formed. By selecting the visualization and pressing the 'm' key, the user can cycle through volume view, line view, and point view. Figure 3.5(a) also shows the region where particles are distributed during the first iteration (Example B). Furthermore, Fig. 3.5(b) shows the boxes used for the specular reflection of particles (Example F).



(a) Boxes representing the simulation domain (green) and the region particles are created (blue).

(b) Boxes for specular reflection are in red.

Fig. 3.5: Figures from World-Rank.html.

Graphical User Interface (In Development)

As TURF runs through "initialization" of class objects, it gathers the input parameters available for each object from the source code. TURF also parses actual inputs from `world.list` and `operations.list` files. With these data, a html file named `index.html` is created, which is essentially a Graphical User Interface (GUI) to generate/modify TURF input files, `world.list` and `operations.list`. TURF has gone through a large number of changes since the first release of the GUI. For that reason, the GUI is currently broken and is subjected to be fixed in the near future.

3.3 Grounded Box: 3D Electrostatic PIC with Multi-Species Particle Data

Samuel Araki and Robert Martin

3.3.1 Introduction

This tutorial demonstrates running a simple 3D electrostatic particle in cell (PIC) case in the Thermophysics Universal Research Framework (TURF). This tutorial assumes familiarity with the simple heatbath tutorial. New users are referred to the tutorial for further explanation. The TURF input files can be located in `tutorial-TURF/TURF-IR_2017a/GroundedBox_MS/ES-PIC`. You should see two files with the `.list` extension, which act as the scripting files for TURF.

The grounded box test case was developed to verify TURF's PIC algorithms with respect to AFRL/RD's ICEPIC particle in cell code running in electrostatic PIC mode [11]. The initial conditions are a uniform unit meter cube of zero velocity protons at a density of 10^{10} m^{-3}. In one octant of the cube, the proton charge is neutralized with 10^{10} m^{-3} electrons initially at stationary. The walls of the cube are set to a fixed 0 Volt potential. The electrons are then accelerated by the field due to the charge of the non-neutralized protons in the remaining 7 octants of the box. The field evolves as the electrons accelerate such that the cloud oscillates and evolves within the box. Particles that hit the edge of the box are assumed to be neutralized and removed from the simulation.

3.3.2 world.list

Running the TURF executable in the working directory will have TURF search for the default script file, `world.list`, and parse it automatically. The first block that defines the `WORLD` is shown below. The options defined in this file should look familiar after completing the heatbath tutorial. In this example, the world is named "ICEPIC-Bench" to denote that it was originally intended to serve as a benchmark verification run against the ICEPIC code. The example uses the `operations.list` operations file to define the simulation algorithm which will be discussed below. The remainder of the world definition sets a global cartesian coordinate system with 2 cm cells along with 2.5 ns time steps up to a final simulation time of 10 μs. The next line defines 10 field variables for charge (`Q`), charge density (`rho`), 3 node-centered electric field components (`En`), the electrostatic potential (`phi`), an auxiliary variable for calculating the residual of the potential during the field solve (`residual_phi`), and proton and electron physical and computational particle counts in cells (`NP_H+`, `NP_e-`, `NC_H+`, and `NC_e-`).

The example run is broken into 4 stages named `INITIALIZE`, `FIELD`, `MOVE`, and `PLOT`. The two additional stages compared to the heatbath example are to accommodate an iterative electrostatic potential solve stage (`FIELD`) and to ensure synchronization prior to the plotting operation stage (`PLOT`), though the latter is not strictly necessary. In this tutorial, the operations file will be considered in stages (Section 3.3.3). List of operations defined in `operations.list` are given in Table 3.6. The last section of `world.list` defines the active domain for the simulation. This example is simply a 1m unit cube starting from the coordinate origin. Using the global mesh spacing of 2 cm from the `WORLD` definition results in a $50 \times 50 \times 50$ active cell cube with the default 3 "ghost"-cells added to the high and low side in each direction for application of boundary conditions.

```
DEFINE WORLD
       NAME = ICEPIC-Bench
       PLOT_FILE = plots.list
       OP_FILE = operations.list
       COORDINATES = cartesian
       ORIGIN = (0.0,0.0,0.0)
       DELTA = (0.02,0.02,0.02)
       START_DT = 2.50e-9
       END_TIME = 10.0e-6
       FIELDS = [Q, rho, Enx, Eny, Enz, phi, residual_phi, NC_e- NC_H+ NP_e- NP_H+ ]
       MATERIALS = [e-, H+@g]
       STAGES = [INITIALIZE, FIELD, MOVE, PLOT]
       START_ITERATION = 0 # Number of Poisson Iteration Before Start
```

```
END WORLD

###########################################################################
## Domain Geometry
###########################################################################
DEFINE DOMAIN DOM000
      BOUND_LO = (0.0,0.0,0.0)
      BOUND_HI = (1.0,1.0,1.0)
END DOMAIN
```

Table 3.6: Summary of operations listed in `operations.list`.

Stage	Operation	Description
INITIALIZE	`MSPDistInitOp`	Create particle distribution
	`MSPDistBoxICOp`	Initialize particle distribution within the domain
	`MSPDistCellIDOp`	Flag cell in which particle resides
	`MSPDistSortOp`	Sort particles in cells by CellID
	`MSPDistSampleOp`	Sum particles to cells
	`FieldSetOp`	Set field values to constant
	`FieldArithmeticCoeffOp`	Compute total charge
	`NextStageOp`	Continue to next stage
FIELD	`LogicalBCConstantOp`	Set value of cell centers in box every iteration
	`LogicalPoissonStripOp`	Red/Black line relaxing Poisson solve
	`LogicalResidualOp`	Calculate residual of Poisson solve
	`LogicalNormOp`	Calculate L^p-norm of field variable
	`CriteriaStageOp`	Continue to next stage if quantity below criteria
MOVE	`LogicalNodeGradientOp`	Calculate node-centered gradient of cell center field
	`MSPDistESPushOp`	Electrostatic particle push using node electric field
PLOT	`VolumeRenderOp`	Single cubic domain realtime volume rendering
	`LogicalFieldWriteVTKOp`	Write to output files for 3D plots

3.3.3 operations.list

Stage: Initialize

This stage creates MSPDist objects that hold electron and proton macro-particles, distributes these particles uniformly within a box, and removes particles that are outside of the simulation domain. The `MSPDistInitOp` and `MSPDistBoxICOp` operations should be familiar from the heatbath example. "Ghost" particle distributions are also created, again using `MSPDistInitOp`. These are empty buffers where particles that have escaped the domain get copied later on in the sort.

```
DEFINE STAGE INITIALIZE

###########################################################################
## Initial Particle Distibutions and Ghost/Exchange Distributions ##
###########################################################################
DEFINE OPERATION
      TYPE = MSPDistInitOp
      MSPDIST_DATA_NAME = P-DST
      MAX_NP = 4000000
      SPECIES_NAMES = e- H+@g
```

```
END OPERATION
DEFINE OPERATION
       TYPE = MSPDistInitOp
       MSPDIST_DATA_NAME = P-GST
       MAX_NP = 4000000
       SPECIES_NAMES = e- H+@g
END OPERATION

################################################
### Distribute Particles in a Box Uniformly ###
################################################
DEFINE OPERATION
       TYPE = MSPDistBoxICOp
       MSPDIST_DATA_NAME = P-DST
       SPECIES = e-
       BOUND_LO = (0.0,0.0,0.0)
       BOUND_HI = (0.5,0.5,0.5)
       TEMPERATURE = 0.0
       NUMBER_DENSITY = 1.0e10
       REAL_TO_COMPUTATIONAL = 1.25e3
       VEL = (0.0,0.0,0.0)
       INIT_ONLY = TRUE
END OPERATION
DEFINE OPERATION
       TYPE = MSPDistBoxICOp
       MSPDIST_DATA_NAME = P-DST
       SPECIES = H+@g
       BOUND_LO = (0.0,0.0,0.0)
       BOUND_HI = (1.0,1.0,1.0)
       TEMPERATURE = 0.0
       NUMBER_DENSITY = 1.0E10
       REAL_TO_COMPUTATIONAL = 8.0e4
       VEL = (0.0,0.0,0.0)
       INIT_ONLY = TRUE
END OPERATION
```

The next set of operations sort the particles by the cell in which they reside. Though still part of the "INITIALIZE" stage, they will run every time the simulation loops back through that stage. The MSPDistCellIDOp operation identifies which cell the particle resides in and saves it to the "CellID" variable within the particle distribution. The MSPDistSortOp operation sorts the particles by their CellIDs and any particle that has escaped the domain gets separated into the ghost distribution, P-GST.

```
########################################
## Remove Particles Leaving the Domain ##
########################################
DEFINE OPERATION
       TYPE = MSPDistCellIDOp
       MSPDIST_DATA_NAME = P-DST
END OPERATION
DEFINE OPERATION
       TYPE = MSPDistSortOp
       NAME = Sort_P-DST
```

```
      MSPDIST_SRC_NAME = P-DST
      MSPDIST_DST_NAME = P-GST
END OPERATION
```

The next section accumulates particle quantities into the cell field variables. The accumulation of field data is performed by `MSPDistSampleOp`. It sets the diagnostic fields for number of computational (`NC_x`) and real (`NP_x`) particles per cell. It is worth noting that these numbers are both raw sums. Before the data can be accumulated, the field variable "Q" (charge) must be cleared using `FieldSetOp`, and then `FieldArithmeticCoeffOp` calculates the total charge by multiplying the real particle counts, `NP_x`, by their respective charges and summing the values.

```
###################
## Sum to Fields ##
###################

DEFINE OPERATION
      TYPE = MSPDistSampleOp
      FIELD_DATA_NAME = FieldData
      MSPDIST_DATA_NAME = P-DST
      FIELD_NAMES = NC_e- NC_H+ NP_e- NP_H+
      MIX_START_TIME = 1.0 # Don't mix
      STEPS_PER_SAMPLING = 1
      STEPS_PER_FIELD_CALCULATION = 1
END OPERATION
DEFINE OPERATION
      TYPE = FieldSetOp
      FIELD_DATA_NAME = FieldData
      FIELD_NAME = Q
      VALUE = 0.0
      OPERATION = SET
END OPERATION
DEFINE OPERATION
      TYPE = FieldArithmeticCoeffOp
      FIELD_DST_NAME = Q
      FIELD_SRCB_NAME = NP_e-
      FIELD_SRCC_NAME = NP_H+
      COEFFICIENT_B = -1.602189200e-19
      COEFFICIENT_C = 1.602189200e-19
      OPERATION = ADD
END OPERATION
```

Finally, the "`INITIALIZE`" stage is completed with the `NextStageOp` operation to proceed to the "`SOLVE`" stage.

```

DEFINE OPERATION
# Default Criteria to Proceed to the Next Stage
      TYPE = NextStageOp
END OPERATION

END STAGE INITIALIZE

#############################################################################
```

Stage: Field

This stage iterates on solving for the electrostatic potential until the residual is small enough to proceed. The first step of the iterative field solve is to set the boundary condition potential to 0 on all six faces of the box. This is done using the `LogicalBCConstantOp` operation. The operation is relatively straightforward. The `phi` variable of the default `FieldData` object is set to a potential of 0 Volts inside of the box defined by `BOUND_LO` and `BOUND_HI`. In this configuration of TURF, the potential is assumed to be cell centered. More specifically, the potential is set to 0 for every cell which has a cell center inside the physically defined box. This may lead to errors on the order of Δx on the location of the application of the boundary condition, but with the boundary conditions defined physically, the solution should converge to the exact solution with grid refinement without manual reconfiguration of the operations. The same approach is used when creating the domains which snap to the nearest approximation of cells based on the physical constraints independent of the underlying mesh resolution. Once again, the `NAME` variable for the operation is simply a designator label for output readability. The value in the `NAME` is not evaluated by the code to influence application of the operation. Boundary condition boxes are chosen to be large enough to contain at a minimum the first few layers of cell centers even at the coarsest resolutions run. In regions where the physical boundary conditions overlap, the value will be set repeatedly.

```
DEFINE STAGE FIELD
DEFINE OPERATION
       TYPE = LogicalBCConstantOp
       NAME = Electrode-X-
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = phi
       BOUND_LO = (-0.1,-0.1,-0.1)
       BOUND_HI = (0.0,1.1,1.1)
       VALUE = 0.0
END OPERATION
DEFINE OPERATION
       TYPE = LogicalBCConstantOp
       NAME = Electrode-X+
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = phi
       BOUND_LO = (1.0,-0.1,-0.1)
       BOUND_HI = (1.1,1.1,1.1)
       VALUE = 0.0
END OPERATION
DEFINE OPERATION
       TYPE = LogicalBCConstantOp
       NAME = Electrode-Y+
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = phi
       BOUND_LO = (-0.1,1.0,-0.1)
       BOUND_HI = (1.1,1.1,1.1)
       VALUE = 0.0
END OPERATION
DEFINE OPERATION
       TYPE = LogicalBCConstantOp
       NAME = Electrode-Y-
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = phi
       BOUND_LO = (-0.1,-0.1,-0.1)
       BOUND_HI = (1.1,0.0,1.1)
       VALUE = 0.0
```

```
END OPERATION
DEFINE OPERATION
       TYPE = LogicalBCConstantOp
       NAME = Electrode-Z+
       VALUE = 0.0
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = phi
       BOUND_LO = (-0.1,-0.1,1.0)
       BOUND_HI = (1.1,1.1,1.1)
       VALUE = 0.0
END OPERATION
DEFINE OPERATION
       TYPE = LogicalBCConstantOp
       NAME = Electrode-Z-
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = phi
       BOUND_LO = (-0.1,-0.1,-0.1)
       BOUND_HI = (1.1,1.1,0.0)
       VALUE = 0.0
END OPERATION
```

After the boundary conditions have been set, the actual potential solver is performed. In this example, `LogicalPoissonStripOp` is used to solve Gauss's Law (the integral form of Poisson's equation).

$$\oint_s \mathbf{E} \cdot \mathbf{dn} = \frac{Q}{\epsilon_0} \tag{3.1}$$

where $\mathbf{E}$ is the electric field given as $\mathbf{E} = \nabla\phi$, ϕ is the electric potential, $\mathbf{n}$ is the normal vector, Q is the charge, and ϵ_0 is the free space permittivity. Equation (3.1) is applied to every cell, and the right hand side of Eq. (3.1) is approximated with finite difference equations. Currently, the set of elliptic solvers in TURF is relatively minimal and includes only red-black Gauss-Seidel and tri-diagonal ADI-type solvers. There is also degenerate 1D version of the solver that can be used in fundamentally 1D problems or as an accelerated initial guess for solutions that are primarily one dimensional. `LogicalPoissonStripOp` also has an option to sub-cycle the calculation multiple times before continuing (`SUBCYCLE`). The operation is applied in a red-black checkerboard in the iterative directions so that the solution is independent of the order in which the line relaxation sweeps are performed. Note that this operation requires the charge `Q` instead of the charge density `rho` as it solves the Gauss's equation instead of the Poisson equation.

```
DEFINE OPERATION
       TYPE = LogicalPoissonStripOp
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = phi
       SOURCE_NAME = Q
       SMESH_NAME = SMesh
       SUBCYCLE = 3
END OPERATION
```

The last part of the `FIELD` stage is defining the criteria to iterate the stage or continue to the next. First, the residual of `phi` is computed in every cell and stored in `residual_phi` using the `LogicalResidualOp`. Next, the `LogicalNormOp` operation calculates the norm of the residual and stores in `NORM_NAME` named as `SUMresidual_phi_L2`. The `NORM` parameter defines the power p for any L^p-norm. Finally, the `CriteriaStageOp` evaluates whether the summed residual is below the required threshold `CRITERIA`. Each domain applies this operation independently. At

the end of each stage, every process collects one vote from every domain as to whether or not to proceed to the next stage or to loop to iterate on the stage. These votes are broadcast across all processes and evaluated by the world when determining whether or not to proceed.

```
DEFINE OPERATION
       TYPE = LogicalResidualOp
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = phi
       SOURCE_NAME = Q
       SMESH_NAME = SMesh
       RESIDUAL_NAME = residual_phi
END OPERATION
DEFINE OPERATION
       TYPE = LogicalNormOp
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = phi
       RESIDUAL_NAME = residual_phi
       NORM = 2.0
       NORM_NAME = SUMresidual_phi_L2
END OPERATION

DEFINE OPERATION
# Default Criteria to Proceed to the Next Stage
       TYPE = CriteriaStageOp
       QUANTITY_NAME = SUMresidual_phi_L2
       CRITERIA = 5.0e-4
END OPERATION

END STAGE FIELD

################################################################################
```

Stage: Move

In this stage, the particle positions are updated using the potential solved in the prior step. To do this, the node centered electric field, `En`, is evaluated first using the `LogicalNodeGradientOp` operator. Because the field is the negative gradient of the potential, the `FIELD_MULTIPLY_CONSTANT` of `-1.0` is included. The `FIELD_GRADIENT_DIRECTIONS` are suffixes attached to the root name `En` that the operator uses to construct the three components of the field names needed to store the result.

```
DEFINE STAGE MOVE

DEFINE OPERATION
       TYPE = LogicalNodeGradientOp
       FIELD_DATA_NAME = FieldData
       FIELD_POTENTIAL_NAME = phi
       FIELD_GRADIENT_PREFIX = En
       FIELD_MULTIPLY_CONSTANT = -1.0
       FIELD_GRADIENT_DIRECTIONS = [x, y, z]
END OPERATION
```

The next operation uses the electric field to advance the electron and proton positions. The inputs are similar to the basic linear push described in the heatbath tutorials, but with extra field options so that the operator knows

which field data to use for the acceleration. This push does not actually test boundary intersections during the push, which is a fast method for simple boundary conditions. After the particle push, the last operation in the stage, `NextStageOp`, tells TURF to proceed to next stage.

```
DEFINE OPERATION
      TYPE = MSPDistESPushOp
      FIELD_DATA_NAME = FieldData
      FIELD_EN_PREFIX = En
      FIELD_EN_DIRECTIONS = [x, y, z]
      MSPDIST_DATA_NAME = P-DST
      FIELD_POTENTIAL_NAME = phi
      SPECIES_NAMES = e- H+@g
END OPERATION

DEFINE OPERATION
      TYPE = NextStageOp
END OPERATION

END STAGE MOVE
##############################################################################
```

Stage: Plot

The last stage of the simulation is plotting. The first operation is the CUDA accelerated real-time volume ray tracing operation, `VolumeRenderOp`. In order to use this operation, TURF has to be built with `USE_CUDA` and `USE_GL` options. The operation is primarily a wrapped version of the NVIDIA CUDA SDK's VolumeRender example. The infrastructure launches that set of code in a separate window. When the operation is applied during the code's main thread loop, a second buffer is filled from the field variable specified by the `FIELD_DATA_NAME` and `FIELD_NAME` parameters. It then signals the visualization thread to swap buffers. It is restricted to single cubic domains in this version of the infrastructure because it uses the rendering kernels from the example with few modifications to apply in other geometries. Most of the settings for producing the coloring and view were obtained by interacting with the visualization to determine a 'good' view. This mode of interaction is described below the file listing. Other options include the `FILE_HEAD` and `SAVE_IMG` options. If enabled, the operation outputs a '.ppm' image file for every iteration that is drawn. Iteration skipping can be adjusted by the `SKIP` parameter to reduce the number of files. The `VIEW_ORBIT` parameter tells the visualization to rotate by the indicated number of degrees once per iteration automatically in addition to the interactive rotations to help make the 3D nature of the volume rendering more intuitive. An example of the output displayed with the default settings by the realtime visualization can be seen in Fig. 3.6. This shows the electron cloud density in the box after 828 timesteps using the default visualization parameters.

```
DEFINE STAGE PLOT

DEFINE OPERATION
      TYPE = VolumeRenderOp
      FIELD_DATA_NAME = FieldData
      FIELD_NAME = Ne-
      SKIP = 1
      DENSITY = 0.04
      BRIGHTNESS = 1.7
      TRANSFERUPPERBOUND = 3.8e5
      TRANSFERLOWERBOUND = 2.5e3
      LOG_PLOT = FALSE
```

```
      INVERT = FALSE
      VIEW_TRANSLATION = (0.0,0.0,-3.6)
      VIEW_ROTATION = (0.4,51.6,0.0)
      VIEW_ORBIT = (0.0,-2.0,0.0)
      WINDOW_SIZE = (960,960)
END OPERATION
```

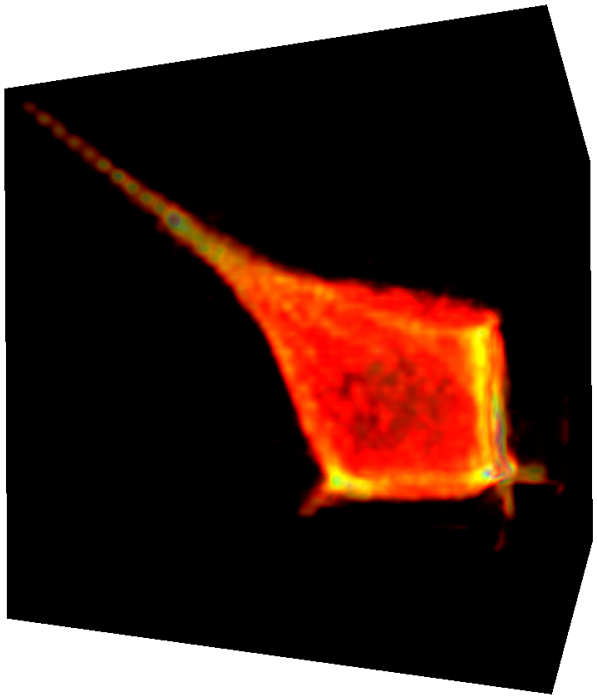


Fig. 3.6: Volume rendering example output of electron density in grounded box

Left-clicking and dragging the mouse rotates the visualization. Right-clicking and dragging the mouse scales the view. Center or simultaneous left and right clicking while dragging the mouse pans the viewport. The '`-+`' keys adjust the density for the ray tracing. Lower values make the electron cloud more translucent and higher makes the rendering thicker and only values closer to the surface of the cloud are visible. The square bracket keys, '`[]`', adjust the 'brightness' of the display. The keys on the next row down, ';'', adjust `transferUpperBound`, which is essentially the top edge of the colormap. The next row down from there, the ',.' keys adjust`transferLowerBound`. This is similarly the bottom edge of the colormap. The '`i`' key inverts the coloring of the display to a black box on a white background. As the keys adjust the settings, the visualizer displays the adjusted parameters interwoven with normal output from the infrastructure. Once a good view has been determined, the options can then be fed back into the operation's parameters for future runs. The output of holding the '`-`' is shown below with some additional whitespace for clarity while a similar line is produced by the mouse adjustments as well.

```
Iteration 1747: Time=4.367510e-06 dt=2.500000e-09 [Wall Clock:477.743864]
density = 0.07, brightness = 2.10, transferUpperBound = 3.45e+05,
       transferLowerBound = 1.39e+04, invert = F
density = 0.06, brightness = 2.10, transferUpperBound = 3.45e+05,
       transferLowerBound = 1.39e+04, invert = F
```

```
density = 0.05, brightness = 2.10, transferUpperBound = 3.45e+05,
       transferLowerBound = 1.39e+04, invert = F
NORM: 4.386121e-04
Iteration 1748: Time=4.370010e-06 dt=2.500000e-09 [Wall Clock:478.055719]
```

The last additional operations are a commented version of the `LogicalFieldWriteVTKOp` which writes the field data to output files rather than relying on the realtime visualization. This is necessary for running the tutorial on systems that do not include NVIDIA GPU's that are compatible with the direct OpenGL interface used by the volume renderer. The options are similar to those of the heatbath tutorial. The last operation is a final `NextStageOp` to tell the code to advance to the next stage, or in this case, loop back to the first stage.

```
# DEFINE OPERATION
# TYPE = LogicalFieldWriteVTKOp
# FIELD_DATA_NAME = FieldData
# FILE_HEAD = Plot3D/plt_
# FIELD_NAMES = Ne- CNe- Np+ CNp+ phi rho
# SKIP = 5
# HELP = TRUE
# END OPERATION

DEFINE OPERATION
      TYPE = NextStageOp
END OPERATION

END STAGE PLOT
################################################################################
```

3.3.4 Results

The example in this tutorial was originally developed to verify TURF functionality with respect to the ICEPIC code. Using as similar parameters as possible between the two codes, the example was run and visualized in ParaView as shown in Fig. 3.7. The setup was nearly identical to what was outlined above except more particles were used to provide smoother output. In particular, particle specific weights of 1.25×10^3 and 1.0×10^4 are used for electron and proton, respectively, to ensure a similar number of particles were used in TURF as in ICEPIC. For the realtime visualization, the low proton numbers make little difference in the electron density visualization, but they make charge density plots like those used to compare the code much more noisy. The agreement between the two codes was very reasonable considering all the particle trajectories are coupled to the field solution and vice versa. A major difference is the appearance of more charge neutrality on the surface of the ICEPIC result, but this is essentially a difference due to node-centered versus cell-centered output between the two codes. On longer timescales after the protons have had the opportunity to move further, the noise level in TURF would appear more similar to that in the ICEPIC result.

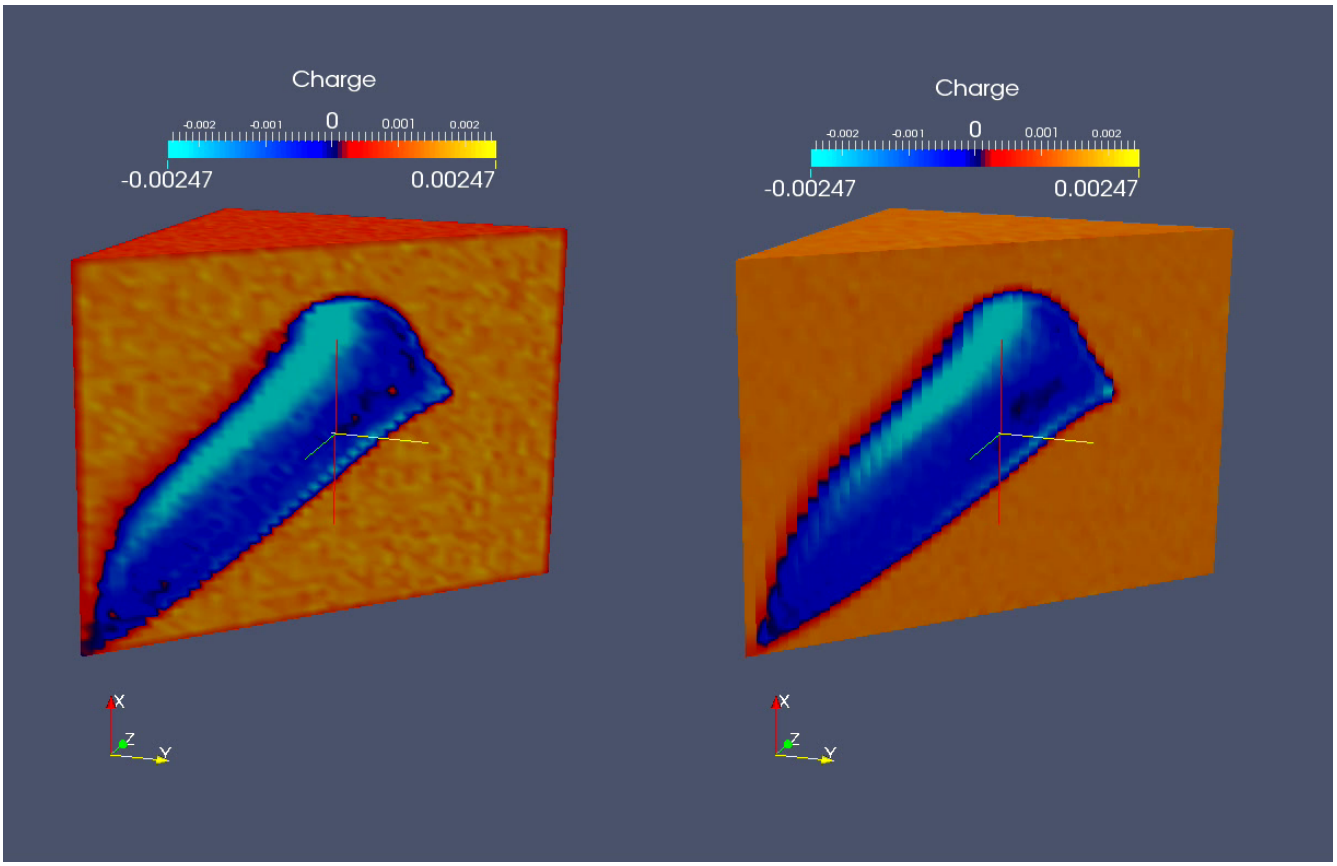


Fig. 3.7: Comparison of ICEPIC (left) and TURF (right) grounded box results

3.4 One Dimensional Normal Shock: DSMC with Multi-Species Particle Data

Samuel J. Araki

3.4.1 Introduction

This tutorial demonstrates running a DSMC case for one-dimensional (1D) normal shock problem in the Thermophysics Universal Research Framework (TURF). In the directory `tutorial-TURF/TURF-IR_2017a/1DShock_MS/DSMC`, you should see two sub-directories named as `DS1V` and `TURF`. The `DS1V` directory contains Bird's DS1V program [14] and input files for four different inflow conditions. The results from the program serve as the reference when verifying the correctness of TURF results. In the `TURF` directory, there should be five files with the `.list` extension, which act as the scripting files for TURF. The script, `world.list`, defines the simulation size, the number of stages, and available materials, etc. This file also points to another script, `operations.Mxx.list`[9], containing all the TURF operations with their inputs. This tutorial first covers the fundamental equation that governs the 1D normal shock problem in Section 3.4.2. Section 3.4.3 then explains how to set up 1D normal shock problem in TURF, providing the global parameters defined in `world.list`. Individual operations defined in `operations.Mxx.list` are described in Section 3.4.4. Finally, the results from TURF and DS1V are compared in Section 3.4.5.

3.4.2 Description of the Example Problem

In a fluid, disturbance information is communicated within a medium at the speed of sound, allowing the upstream flow field to adjust accordingly. However, when the flow velocity is greater then the speed of sound, the disturbance information cannot be communicated fast enough, resulting in a formation of a shock. The shock creates a "discontinuity" or a sudden change in flow properties such as velocity, pressure, and temperature. Across a shock, the pressure and temperature always increase while the velocity always decreases from upstream to downstream. The example to simulate with the DSMC module of TURF is the 1D normal shock problem, in which the shock forms in a plane perpendicular to the flow direction. In this problem, the flow properties at upstream and downstream regions with respect to the shock location are related through the following equations [12].

$$\begin{aligned} \rho_1 u_1 &= \rho_2 u_2 \\ p_1 + \rho_1 u_1^2 &= p_2 + \rho_2 u_2^2 \\ h_1 + \frac{1}{2}u_1^2 &= h_2 + \frac{1}{2}u_2^2 \end{aligned} \tag{3.2}$$

where ρ is the density, u is the velocity, p is the pressure, h is the enthalpy, and subscripts 1 and 2 denote upstream and downstream, respectively. Equation (3.2) is obtained by integrating the Euler equations, a set of conservation equations for mass, momentum, and energy that are applicable for such flows [13]. In a perfect gas, the speed of sound, a, can be determined using the isentropic relation.

$$a^2 = \left(\frac{\partial p}{\partial \rho}\right)_s = \frac{\gamma p}{\rho} = \gamma R_s T \tag{3.3}$$

where γ is the heat capacity ratio defined as $\gamma = 1 + 2/f$, f is the degree of freedom (i.e. 3 for a monatomic gas and 5 for a diatomic gas), R_s is the specific gas constant (i.e. 208.13 J/kg·K for argon), and T is the temperature. Using Eq. (3.2) and the perfect gas assumption, the downstream flow properties can be determined if the upstream flow properties are known [12].

$$M_2^2 = \frac{1 + \frac{\gamma-1}{2}M_1^2}{\gamma M_1^2 - \frac{\gamma-1}{2}} \tag{3.4}$$

$$\frac{n_2}{n_1} = \frac{\rho_2}{\rho_1} = \frac{u_1}{u_2} = \frac{(\gamma+1)M_1^2}{(\gamma-1)M_1^2 + 2} \tag{3.5}$$

$$\frac{T_2}{T_1} = 1 + \frac{2(\gamma-1)}{(\gamma+1)^2}\frac{\gamma M_1^2 + 1}{M_1^2}\left(M_1^2 - 1\right) \tag{3.6}$$

[9] A number corresponding to a different inflow Mach number is used in place of "xx".

where M is the Mach number defined as $M = u/a$ and n is the number density. In setting up the 1D normal shock problem, the downstream flow properties need to be evaluated and input in `operations.Mxx.list` prior to running TURF.

3.4.3 Setting up the DSMC Example

One way to set up the 1D normal shock problem is to introduce uniformly distributed gases upstream and downstream of the shock location. Given the upstream flow properties, appropriate downstream flow properties are determined by Eqs. (3.4) to (3.6). Table 3.7 provides the downstream flow properties for argon gases of $T_1 = 293$, $n_1 = 1 \times 10^{22}$ m^{-3}, and $a_1 = 318.8$ m/s at M_1 of 1.2, 1.4, 2.0, and 8.0. The upstream flow velocities corresponding to the Mach number of 1.2, 1.4, 2.0, and 8.0 are 382.4, 446.2, 637.4, and 2549.6 m/s, respectively. In order to maintain the gas density and the shock location, the gas should also be flowing into the domain from the upstream boundary according to the flow 1. At the interface between the two gases at different flow properties, the properties are initially discontinuous, and they will develop smooth profiles as time evolves. These profiles can be compared with the profiles obtained by other DSMC models or fluid models. Examples of shock profiles are also provided in Ref. [14] (Chapter 12).

The script file `world.list` (shown below) includes global parameters that define the problem such as computational grid size, time-step, species, etc. In TURF, the number of grid cells is determined by finding how many `DELTA`s fit in the user-defined domain bounds (`BOUND_HI-BOUND_LO`), and the domain bounds are redefined such that the coordinates of the lowest corner of the domain box are placed at `ORIGIN`. Therefore, the domain box may not necessarily be bounded by the user-defined `BOUND_LO` and `BOUND_HI`. In this example, the grid contains 1000 cells in x-direction and a single cell for both the y- and z-directions. Here, `BOUND_HI` is set to slightly larger numbers than intended; this ensures that the domain size is not reduced by one cell due to the rounding down of the number of cells. Only argon gas (`Ar@g`) is used in this simulation, and the field parameters to be computed are `NC`, `NP`, `N`, `VX`, and `T` as defined in Table 3.8. This example uses three stages (`INITIALIZE`, `MOVE`, and `PLOT`), and the script file `operations.Mxx.list` contains all the operations within each of the three stages, as listed in Table 3.9. It should be noted that the field and stage names are not case sensitive.

```
DEFINE WORLD
       NAME = DSMC_example
       OP_FILE = operations.Mxx.list
       COORDINATES = cartesian
       ORIGIN = (0.0,0.0,0.0)
       DELTA = (2.0e-5,2.0e-3,2.0e-3)
       END_TIME = 8.00001e-4
       START_DT = 1.0e-8
       FIELDS = [NC NP N VX T]
       MATERIALS = [Ar@g]
       STAGES = [INITIALIZE, MOVE, PLOT]
       PRINT_PROFILE_INFO = true
       SAVE_PROFILE_INFO = true
END WORLD

###############################################################################
## Domain Geometry
###############################################################################
DEFINE DOMAIN DOM000
       BOUND_LO = (0.0,0.0,0.0)
       BOUND_HI = (2.00001e-2,2.00001e-3,2.00001e-3)
END DOMAIN
```

Table 3.7: Downstream flow properties for upstream Mach number of 1.2, 1.4, 2.0, and 8.0. The values are for argon gas.

Downstream			Upstream Mach Number, M_1			
Flow Property	Symbol	Unit	1.2	1.4	2.0	8.0
Velocity	u_2	m/s	294.9	282.3	278.9	667.3
Speed of Sound	a_2	m/s	348.4	376.0	459.4	1456
Mach Number	M_2	-	0.85	0.75	0.61	0.46
Number Density	n_2	$1/m^3$	1.30×10^{22}	1.58×10^{22}	2.29×10^{22}	3.82×10^{22}
Temperature	T_2	K	350.1	407.8	608.9	6116

Table 3.8: Volume mesh fields[a] computed in this DSMC example.

Field Name	Description
NC	Number of simulation particles.
NP	Number of physical particles.
N	Density.
V	Velocity. VX, VY, VZ are the x-, y-, and z-components.
T	Temperature. TX, TY, TZ are the x-, y-, and z-components.

[a] For multi-species simulations, field data for each species can be obtained by adding species name followed by a underscore. For example, if there are two species A@g and B@g, partial densities for species A@g and B@g and total density are named as N_A@g, N_B@g, and N, respectively.

Table 3.9: List of operations in `operations.Mxx.list`.

Stage	Operation	Description
INITIALIZE	`MSPDistInitOp`	Create MSPDist object that contains particle distribution
	`MSPDistBoxICOp`	Fill particle inside a box
MOVEOP	`MSPDistNormalMaxwellianOp`	Injection of particles at Maxwellian distribution
	`MSPDistMoveOp`	Advancement of particles
	`MSPDistBCSpecOp`	Specular boundary condition
	`MSPDistCellIDOp`	Find cell ID associated with particle location
	`MSPDistSortOp`	Sort particles according to cell ID and remove particles outside the simulation domain
	`MSPDistDSMCOp`	DSMC collision calculation
	`MSPDistSampleOp`	Sample particles and compute volume mesh field data
POSTOP	`LogicalFieldWriteVTKOp`	Output to a vtk file for a 3D plot
	`LogicalFieldWrite1DOp`	Output to a csv file for a line plot
	`MSPDistParticleCountOp`	Output particle count

3.4.4 Operations

Stage: Initialize

In this stage, MSPDistInitOp creates a MSPDist object containing particle information, and MSPDistBoxICOp uniformly distributes particles within a box placed inside the simulation. The buffer size of the MSPDist object is specified by MAX_NP in MSPDistInitOp. Users must be careful when choosing a value for MAX_NP; if the particle count exceeds the buffer size, TURF may crush with a segmentation error, although we try to add warning/error messages as much as possible when this situation could happen. In addition, SPECIES_NAMES must be the ones already defined for MATERIAL in world.list. In this example, two MSPDist particle distributions named as P-DST and P-GST are created. P-GST is a temporary MSPDist object that is used when deleting particles in a later stage.

```
DEFINE OPERATION
       TYPE = MSPDistInitOp
       MSPDIST_DATA_NAME = P-DST
       MAX_NP = 1280000
       SPECIES_NAMES = Ar@g
END OPERATION
```

The operation, MSPDistBoxICOp, then adds particles into P-DST. An example of inputs for MSPDistBoxICOp is shown below. The box to be filled with particles is bounded by BOUND_LO and BOUND_HI, in which the lower and higher bounds in Cartesian coordinate are specified, respectively. In this operation, only one gas species can be added at a time, and for this particular case, the gas species to be added is argon. The real to computational particle weight ratio is defined by FNUM. Furthermore, TEMPERATURE, NUMBER_DENSITY, and VEL are the temperature of the gas species, density, and the streaming velocity of the particles at Maxwellian distribution, respectively. In order to set up the 1D shock problem properly, the upstream and downstream regions inside the computational domain are filled with particles according to flow properties 1 and 2. The initial density distribution obtained by the DSMC example is shown in Fig. 3.8. This example corresponds to the case with $M_1 = 1.2$, where the downstream flow properties are given in Table 3.7.

```
DEFINE OPERATION
       TYPE = MSPDistBoxICOp
       MSPDIST_DATA_NAME = P-DST
       SPECIES = Ar@g
       BOUND_LO = (0.0,0.0,0.0)
       BOUND_HI = (1.0e-2,2.0e-3,2.0e-3)
       TEMPERATURE = 293.0
       NUMBER_DENSITY = 1.0e22
       FNUM = 9.1892e8
       VEL = (382.447,0.0,0.0)
       INIT_ONLY = TRUE
END OPERATION
```

Stage: Move

The MOVE stage includes several operations including MSPDistNormalMaxwellianOp, MSPDistMoveOp, MSPDistBC-SpecOp, MSPDistCellIDOp, MSPDistSortOp, MSPDistDSMCOp, and MSPDistSampleOp. MSPDistNormalMaxwellianOp injects particles at Maxwellian distribution from a virtual surface. If the surface mesh name (SURFACE_NAME) and component name (SURFACE_GROUP_NAME) are specified in this operation, the particles are injected from the surface mesh elements. Otherwise, the virtual surface is defined by its center (SOURCE_SURFACE_CENTER), normal vector (SOURCE_NORMAL_VECTOR), tangent vector (SOURCE_TANGENT_VECTOR), length (SOURCE_SURFACE_LENGTH), and type (SOURCE_SURFACE_TYPE). For the surface type, only "SQUARE" and "CIRCLE" surfaces are implemented in the current version. Other input parameters are similar to the ones in MSPDistBoxICOp.

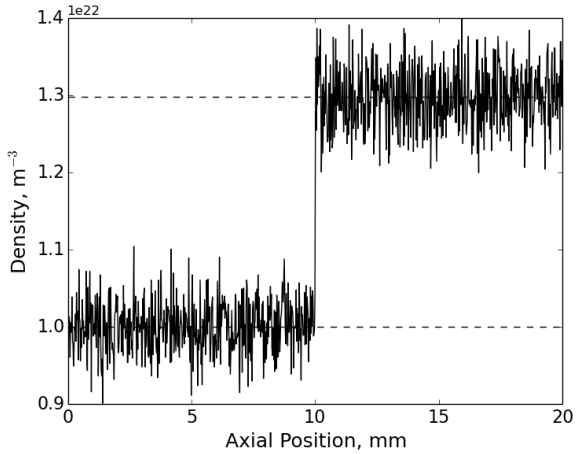


Fig. 3.8: Initial density distribution.

```
DEFINE OPERATION
      TYPE = MSPDistNormalMaxwellianOp
      MSPDIST_DATA_NAME = P-DST
      SPECIES = Ar@g # Can only inject one species
      TEMPERATURE = 293.0
      NORMAL_VELOCITY = 382.447
      NUMBER_DENSITY = 1.0E22
      REAL_TO_COMPUTATIONAL = 9.1892e8
      SOURCE_SURFACE_CENTER = ( 0.0, 1.0e-3, 1.0e-3 )
      SOURCE_NORMAL_VECTOR = ( 1.0, 0.0, 0.0 )
      SOURCE_TANGENT_VECTOR = ( 0.0, 1.0, 0.0 )
      SOURCE_SURFACE_LENGTH = 2.0e-3
      SOURCE_SURFACE_TYPE = SQUARE
      VERBOSE = FALSE
END OPERATION
```

`MSPDistMoveOp` simply advances particles by one time-step. In this operations, only the species defined by `SPECIES_NAMES` in `P_DST` is advanced; in this particular example, only one species exist, which is argon gas.

```
DEFINE OPERATION
      TYPE = MSPDistMoveOp
      MSPDIST_DATA_NAME = P-DST
      SPECIES_NAMES = Ar@g
END OPERATION
```

`MSPDistBCSpecOp` applies specular boundary conditions to the particles. This operation uses a simple operation that utilizes a box defined by `BOUND_LO` and `BOUND_HI`. The specular reflection is applied to particles that lie within the box, and particle positions are altered based on the box surface corresponding to the direction specified in `DIRECTION`. The directions that can be specified are `xm`, `xp`, `ym`, `yp`, `zm`, and `zp`, and the 'm' and 'p' in these directions refer to "minus" (from negative to positive direction) and "plus" (vise versa), respectively. Since the operator is applied to only the particles within the box, it is important to use a box size large enough to capture faster particles.

```
DEFINE OPERATION
       TYPE = MSPDistBCSpecOp
       MSPDIST_DATA_NAME = P-DST
       DIRECTION = ym
       BOUND_LO = (-0.1e-2,-1.0e-3,-1.0e-3)
       BOUND_HI = ( 2.1e-2, 0.0e-3, 3.0e-3)
END OPERATION
```

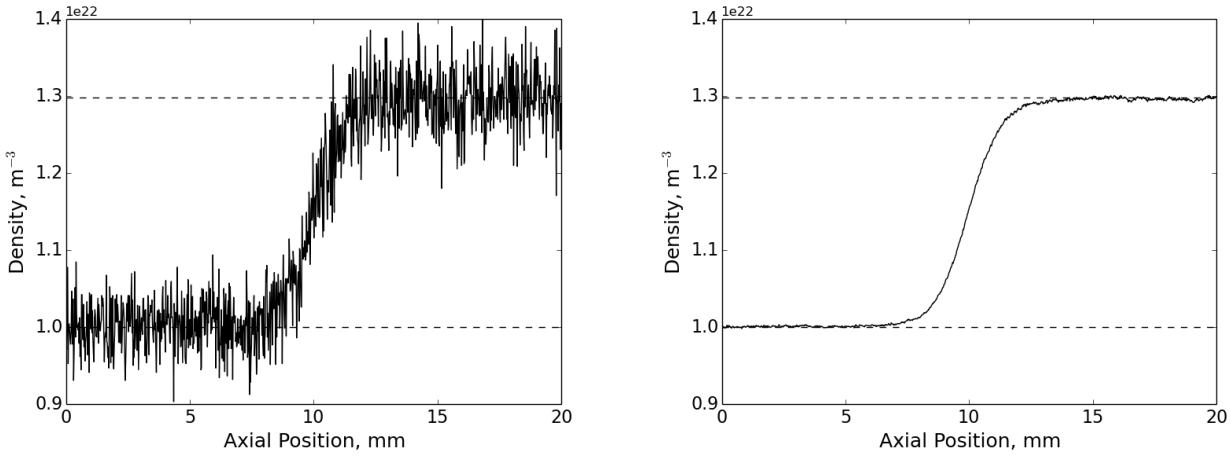
The particles leaving the simulation domain are removed by the combination of two operations, `MSPDistCellIDOp` and `MSPDistSortOp`. `MSPDistCellIDOp` determines the volume mesh cell ID based on where a particle resides. Then, `MSPDistSortOp` sorts the particle distribution in the order of cell ID. For the particles outside the simulation domain, particle cell ID is set to the maximum value, and these particles are copied to `P-GST` in `MSPDistSortOp` and thrown away later. Although removing the particle outside the simulation domain can be done by `MSPDistRemovePartOp`, `MSPDistSortOp` is used instead since the additional operation of sorting the particle distribution makes the DSMC algorithm more efficient.

```
DEFINE OPERATION
       TYPE = MSPDistCellIDOp
       MSPDIST_DATA_NAME = P-DST
END OPERATION
DEFINE OPERATION
       TYPE = MSPDistSortOp
       NAME = Sort_P-DST
       MSPDIST_SRC_NAME = P-DST
       MSPDIST_DST_NAME = P-GST
END OPERATION
```

`MSPDistDSMCOp` finds the number of collisions to perform within all the grid cells and applies the DSMC method. Compared to the Monte Carlo Collision (MCC) method, the DSMC method performs a detailed balance collision in a way that the overall momentum is conserved within the whole system. The variable hard sphere (VHS) molecular model is used to determine the deflection of particles, and the VHS parameters are defined in `materials.list`. The VHS parameters include the reference temperature (T_{ref}), an empirical factor for the variable cross-section (α), and the reference diameter (d), and these parameters can be found in Ref. [14]. For the details of the DSMC method, readers should refer to Ref. [14].

```
DEFINE OPERATION
       TYPE = MSPDistDSMCOp
       MSPDIST_DATA_NAME = P-DST
       SPECIES_NAMES = Ar@g
       FNUM = 9.1892e8
       INIT_TEMPERATURE = 350.074
       FREQUENCY_TO_RESAMPLE_MFS = 2000
       SORT_OP_NAME = Sort_P-DST
END OPERATION
```

`MSPDistSampleOp` samples mass, momentum, and kinetic energy from a particle distribution and computes volume mesh field data from these sampled quantities. The quantity to be sampled depends on `FIELD_NAMES` defined in the input. For example, if the temperature field `T` does not need to be determined, the kinetic energy is not sampled from the particle distribution. In this way, the time to perform the sampling calculation can be reduced when less field data are only needed. The frequencies of sampling and field calculation are defined by `STEPS_PER_SAMPLING`



(a) 8.0×10^{-5} sec, 8,000 Steps

(b) 2.0×10^{-4} sec, 20,000 Steps

Fig. 3.9: Density distribution at different time-steps. Mixing is started at 8,000th time-step.

and STEPS_PER_FIELD_CALCULATION. STEPS_PER_SAMPLING is typically set to a value greater than one to remove the correlation of the flow field caused by numerics (e.g. particle injection). STEPS_PER_FIELD_CALCULATION is typically set to be the same as the output frequency defined by SKIP in operations such as LogicalFieldWriteVTKOp and LogicalFieldWrite1DOp. After the time defined by MIX_START_TIME, sampled quantities are mixed between iterations in order to improve the statistics. Therefore, this operation allows a smooth field distribution at the end of simulation without using a very large number of simulation particles. Here, MIX_START_TIME should be set to the time when the simulation becomes steady-state. The field distributions at 8,000 and 20,000 time-steps are shown in Figs. 3.9(a) and 3.9(b), respectively. In this example, mixing has been performed after 8,000 time-steps; the distribution is smoothed out significantly after mixing the field parameters for the last 12,000 iterations.

```
DEFINE OPERATION
       TYPE = MSPDistSampleOp
       FIELD_DATA_NAME = FieldData
       MSPDIST_DATA_NAME = P-DST
       FIELD_NAMES = NC NP N VX T
       MIX_START_TIME = 8.0e-5
       STEPS_PER_SAMPLING = 5
       STEPS_PER_FIELD_CALCULATION = 100
END OPERATION
```

Stage: Plot

In this tutorial, two different operations to output field data are used. LogicalFieldWriteVTKOp writes field data in vtk format such that the data can be loaded in visualization software like Paraview and VisIt. LogicalFieldWrite1DOp writes 1D field data along x-axis in a csv file to simplify the data extraction for post-processing. For both of these operators, the plotting frequency can be defined by SKIP. LogicalFieldWriteVTKOp has an additional input parameter, PLOT_GHOST, providing users with options to plot the field data within ghost layers.

```
DEFINE OPERATION
       TYPE = LogicalFieldWriteVTKOp
       FIELD_DATA_NAME = FieldData
```

```
      FILE_HEAD = Plot3D_M12/plt_
      FIELD_NAMES = NC NP N VX T
      PLOT_GHOST = FALSE
      SKIP = 100
END OPERATION
DEFINE OPERATION
      TYPE = LogicalFieldWrite1DOp
      FIELD_DATA_NAME = FieldData
      FILE_HEAD = Plot1D_M12/plt_
      FIELD_NAMES = NC NP N VX T
      SKIP = 100
END OPERATION
```

The last operator is unnecessary to run this simulation but is very useful in setting up the simulation. `MSPDistParticleCountOp` may print the number of particles per species and domain to the terminal (`VERBOSE=TRUE`) and/or csv file (`LOG_FILE`). If the particle count reaches nearly constant between iterations, then the simulation has reached steady-state. Otherwise, necessary operations may be missing, or there may be a bug in an operation. If `GLOBAL` is set to true, TURF prints out the total number of particles across all the sub-domains for a multi-domain simulation.

```
      TYPE = MSPDistParticleCountOp
      MSPDIST_DATA_NAME = P-DST
      QUANTITY_NAME = ParticleCount
      LOG_FILE = ParticleCount/M12.csv
      VERBOSE = FALSE
      GLOBAL = TRUE
END OPERATION
```

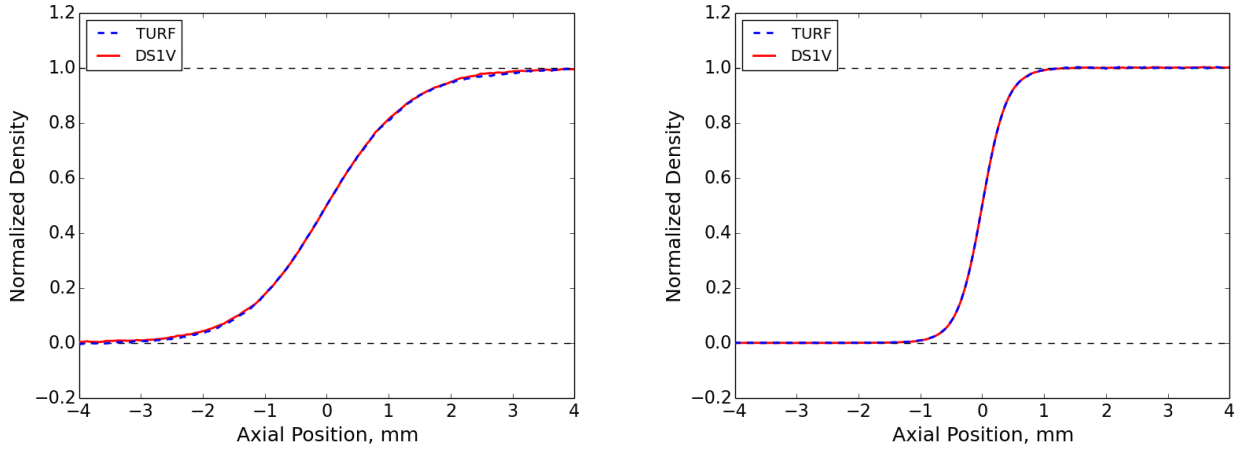
3.4.5 Comparison of Shock Profiles with Bird's DSMC Code

The 1D shock profiles from the DSMC module of TURF can be compared with other DSMC programs for code verification; in this tutorial, the results are compared with the 1D DSMC code, DS1V, developed by G. A. Bird (available at www.gab.com.au). The DS1V source code along with input files (ds1vd.dat) for the cases with M_1 =1.2, 1.4, 2.0, and 8.0 are also provided in this tutorial. In order to obtain a smooth distribution at the end of simulaiton, DS1V is run twice; first, the simulation is started using the "new run" (#3) option in the terminal, the simulation is then stopped at a time greater than 2×10^{-5} sec, and finally the simulation is restarted using "new sample" (#2) followed by the "adapt the cells" (#1) option. The resulting profile of density as a function of position can be extracted from the output file, "PROFILE.DAT."

Figure 3.10 compares the shock profiles computed with TURF and DS1V for the upstream Mach number of 1.2 and 2.0 at 8.0×10^{-4} sec (80,000 time-steps in TURF). The values are normalized according to,

$$\tilde{n} = \frac{n - n_1}{n_2 - n_1} \tag{3.7}$$

It is readily seen from Fig. 3.10 that the agreement between the two programs is quite satisfactory.



(a) M=1.2 (b) M=2.0

Fig. 3.10: Normalized density computed by TURF and DS1V.

3.5 Electric Propulsion Plume Simulation 1: Setup

Samuel J. Araki and Kari A. Kawashima

3.5.1 Introduction

This tutorial can be used as a guide to set up an electric plume (EP) simulation. In the directory `tutorial-TURF/TURF-IR_2017a/Plume/PlumeExample`, there are six files necessary to run this example, and these files are briefly described in Table 3.10. The directory also contains two additional files, `world.restart.list` and `operations.restart.list`, to demonstrate the use of TURF restart capability. TURF uses a tree-hierarchy structure which has the World object at the top level for each MPI process. The World file (`world.sat.list`) contains the global parameters that define the World as well as the Domain object attached underneath the World object. Here, each World object can hold one or more sub-domains or Domain objects. Individual operators defined in `operations.sat.list` are attached under the Domain object such that all of these operators are applied to each domain. A simplified list of steps to set up a new simulation is provided below:

1. Get or create a CAD drawing of a spacecraft.
2. Mesh the geometry and export in an abaqus file format.
3. Modify highlighted entries in `world.list`.
4. Update `component_TURF.txt`. Use component names from the surface mesh file and specify material and potential for each component (if they are different from default values).
5. Identify relevant surface interaction and sputtering mechanisms, obtain coefficients for surface interaction models, and include in `surf_int.txt`.
6. Modify highlighted entries in `operations.sat.list`.
7. Run TURF simulation
8. Evaluate results.

This tutorial consists of three parts. The first part (this document) covers the operations available in TURF for a plume simulation and demonstrates how to alter the input files to suit the needs of various simulations, which corresponds to Steps 3 to 6 in the list above. Part Two of this tutorial demonstrates how to create a surface mesh file using Cubit [21] (Steps 1 and 2), and Part Three covers the ParaView [19] functions to visualize TURF outputs (Step 8). In order to run a single MPI process simulation with TURF (Step 7), a symbolic link to `world.list` needs to be created first.

```
tutorial-TURF/TURF-IR_2017a/Plume/PlumeExample> ln -s world.sat.list world.list
```

TURF always looks for a file named as `world.list`, which is routed to `world.sat.list` by the command shown above. If a symbolic link to `world.list` already exists in the directory, `-sf` option can be used to force replacing the existing link. Once `world.list` is linked to the proper file, TURF can be run by simply typing the name of the binary file including its absolute or relative path from the folder where the input files are located.

```
tutorial-TURF/TURF-IR_2017a/Plume/PlumeExample> ./../../../../bin/TURF-o
```

You can also add a line in `.bashrc` for a search path such that the path does not need be included.

Table 3.10: Description of files in this example.

File	Description
`world.sat.list`	TURF script file containing global parameters defining the entire simulation.
`operations.sat.list`	TURF script file containing individual operations and inputs.
`jaysat2.inp`	Surface mesh file.
`component_TURF.txt`	Parameters associated with surface mesh components.
`surf_int.txt`	Parameters for particle-surface interaction calculations.
`RPA_HCT1.csv`	RPA probe data used for ion source.

```
export PATH = $PATH:"/Path_To_TURF_Binary/bin"
```

Multi-domain simulations can be performed by redefining the domain in `world.list` and running the TURF binary with `mpirun`. Refer to the Heatbath example for setting up and running the two domain simulation. This document first explains the `World` file, and then provides the details of the component (Section 3.5.3) and surface interaction files (Section 3.5.4), followed by a thorough description of the TURF operations and variables (Section 3.5.5).

3.5.2 Defining the World

In order to establish the starting parameters and conditions of the simulation, the TURF calls the World file named as `world.list`. This file contains the name of the simulation, points to an input file that contains the operators, the global parameters of the simulation, etc, and these are assigned to the World object in TURF. Highlighted below are variables in the world file which are most commonly modified for different simulations. Note that the parameter names can either be lower or upper case.

```
DEFINE WORLD
       NAME = plume_example
       OP_FILE = operations.sat.list
       COORDINATES = cartesian
       ORIGIN = ( 0.0, 0.0, 0.0 )
       DELTA = ( 0.25, 0.25, 0.25 )
       END_TIME = 4.00001e-3
       START_DT = 2.0e-6
       FIELDS = [ pflag, rho, rho_avg, phi, Enx, Eny, Enz]
       FIELDS = [ V, V_Xe, V_Xe+, V_Xe+2, VX, VY, VZ, T, NP, NC]
       FIELDS = [ VOL, n, n_Xe, n_Xe+, n_Al, n_Fe, bc ]
       MATERIALS = [Xe@g Xe+@g Al@g Fe@g Al@s Fe@s Cu@s]
       STAGES = [INITIALIZE, FIELD, MOVE, POSTMOVE]
       PRINT_PROFILE_INFO = true
       SAVE_PROFILE_INFO = true
END WORLD

###############################################################################
## Domain Geometry
###############################################################################
DEFINE DOMAIN DOM
       BOUND_LO = (-2.5, -2.5, -2.5)
       BOUND_HI = ( 5.0, 2.5, 5.5)
       SUB_DOMAINS = (1,1,1)
END DOMAIN
```

- `NAME` creates a label for the simulation world.
- `OP_FILE` is the file containing all the individual operations.
- `ORIGIN` is the coordinates of node with index $i = 0$, $j = 0$, and $k = 0$.
- `DELTA` determines the size of a cell in the simulation's domain.
- `END_TIME` and `START_DT` dictate the point at which the simulation stops and the time interval for which the simulation advances, respectively.[10]
- `FIELD_NAMES` names and creates different data sets which will be collected and saved during the simulation.

[10] These values are determined by particle velocity, cell size, and the time required for the particles to reach steady-state. The simulation ends when the TURF time becomes greater than or equal to `END_TIME`. TIP: Use a slightly larger `END_TIME` to account for possible round-off and ensure the simulation completes the last desired step.

- `MATERIALS` establishes all the material types that will be used for the simulation.[11]
- `STAGES` assigns names to the blocks of operations of TURF simulation.[12]
- `BOUND_LO` and `BOUND_HI` set the lower and upper bounds of the simulation's domain.
- `SUB_DOMAINS` partitions the simulation region into multiple sub-domains.[13]

Table 3.11: Volume mesh fields[a,b,c] computed in this plume example.

Field Name	Description
pFlag	Flag to indicated which potential solver to use.
rho	Charge density.
rho_avg	Time-averaged charge density.
phi	Electric potential.
Enx, Eny, Enz	Components of node-centered electric field in x-, y-, and z-directions.
V	Flow speed.
VX, VY, VZ	Components of flow velocity in x-, y-, and z-directions.
T	Temperature.
NC	Number of simulation particles.
NP	Number of physical particles.
n	Density.
VOL	Cell volume.
bc	Sugarcubing info primarily used for debugging purpose.

[a] Field data for each species can be obtained by adding species name followed by a underscore. For example, if there are two species A@g and B@g, partial densities for species A@g and B@g and total density are named as N_A@g, N_B@g, and N, respectively.
[b] FIELD_NAMES are not case sensitive.
[c] All fields are cell-centered otherwise noted.

3.5.3 Geometry Components

The parameters associated with each component of the spacecraft geometry can be specified in a component file (named as `component_TURF.txt` in this example). The file contains a list of components with associated parameters in a JavaScript Object Notation (JSON) like format.

```
component{name:SC_BODY, material:Fe@s, potential:0.0, temperature:300.0}
```

The JSON format is chosen to retain the flexibility as TURF capability is expanded. In this format, there are multiple pairs of keys and values separated by colon associated with a component. The keys are used within TURF operations to access the data when needed, and there is no distinction between upper or lower case for the keys. On the other had, the values of different data type are stored as a TURF string object[14], and they are converted to integer, float, double, or string data type when they are accessed within TURF. The values as string format have to match exactly when referring to them in other parts of inputs; it distinguishes lower and upper case letters.

In the component file, the current version takes the name of the component (`name`) and associated `material`, `potential`, and `temperature`. The `material` and `temperature` are used when performing the surface interaction calculations and `potential` is used as Dirichlet boundary condition for the potential calculations within free-space. The file must use the `name`s given in the surface mesh file; the names also corresponds to the "blocks" defined in Cubit when generating the surface mesh file. If the names do not match exactly, the component specific inputs will

[11] "@g", "@l", and "@s" denote gas particles, liquids, and solids, respectively. The names defined here must coincide with the material "name" or "composition" defined in `src-TURF/src/Materials/database/materials.list`.
[12] MPI communication is performed between these stages.
[13] Each sub-domain can be assigned to different MPI processes for parallel computing.
[14] Stored as GSPrime<string> such that it can be attached to a component object.

not be read, and the default values established in the Initialization stage will be assigned. Similarly, the specified material must be one listed in the `materials` field in the World file.

3.5.4 Particle-Surface Interaction

The surface interaction file, `surf_int.txt`, informs the simulation what to do when gas particles come into contact with the surface of the spacecraft. Both the surface reflection and sputtering are integrated in the current version of TURF. The incident particles can stick to the surface or reflect off the surface specularly or diffusely. These particles can also cause sputtering of surface material. The surface interactions are defined in `surf_int.txt` in the JSON like format also used for `component_TURF.txt`.

```
surface_impact{source:Xe@g, target:Al@s, product:Xe@g, spwt_ratio:1.0, c_stick:1.0, c_rest:1.0,
c_accom:0.9, c_diff:1.0}
```

```
sputtering{model:yama, source:Xe@g, target:Al@s, product:Al@g, spwt_ratio:0.005, sput_c0:3.39,
sput_c1:1.0, sput_c2:2.17, sput_c3:2.5, sput_c4:1.8, sput_c5:2.56, emission:zhang,
emit_c0:72.25, emit_v:5000}
```

For both the surface reflection and sputtering, `source`, `target`, and `product` materials are specified. The specific weight ratio (`spwt_ratio`) is the ratio of product to source particle weight, where the particle weight is the ratio of physical to computational particles.

For surface reflection, four coefficients are defined; C_{stick}, C_{rest}, C_{accom}, and C_{diff} are coefficients of sticking, restitution, thermal accommodation, and diffuse reflection, respectively. If $U < C_{\text{stick}}$ where U is a random number between 0 and 1, then the particle is killed and is no longer tracked. Otherwise, the particle is reflected off the surface. The rest of coefficients are used to smoothly mix the velocity direction ($\hat{\mathbf{v}}$) and magnitude (v) between specular and diffuse reflections.

$$\hat{\mathbf{v}} = \hat{\mathbf{v}}_{\text{spec}} + C_{\text{diff}}(\hat{\mathbf{v}}_{\text{diff}} - \hat{\mathbf{v}}_{\text{spec}}) \tag{3.8}$$

$$v = C_{\text{rest}} v_{\text{spec}} + C_{\text{accom}}(v_{\text{diff}} - C_{\text{rest}} v_{\text{spec}}) \tag{3.9}$$

Here, the velocity due to the diffuse reflection, $\hat{\mathbf{v}}_{\text{diff}}$, is computed based on the wall temperature and by the Box-Muller algorithm.

When adding sputtering for different combination of materials, a sputtering `model` and an `emission` models are required in `surf_int.txt`. In the current version of TURF, seven models for total sputter yield and two models for angular distribution of sputtered particles are implemented and are listed in Tables 3.12 and 3.13 along with the coefficients required to be defined in `surf_int.txt`. The equations for total yield and angular distribution are provided in Sections 3.5.7 and 3.5.8.

Table 3.12: Sputtering models incorporated in TURF. Section 3.5.7 shows how the coefficients are used in total yield calculation.

Model	Key	Required Coefficients	Reference
`Constant`	CONST	C_0	
`Matsunami`	MATSU	C_0 and C_1	[22], [23]
`Yamamura`	YAMA	C_0, C_1, C_2, C_3, C_4, and C_5	[24], [25]
`Kannenberg`	KANNEN	C_0, C_1, C_2, C_3, and C_4	[26]
`Roussel`	ROUSSEL	C_0 and C_1	[27]
`Garnier`	GARNIER	C_0, C_1, C_2, C_3, C_4, and C_5	[28]
`Pencil`	PENCIL	C_0 and C_1	[29]

Table 3.13: Emission models incorporated in TURF. Section 3.5.8 shows how the coefficients are used in emission angular distribution calculation.

Model	Key	Required Coefficients	Reference
`Cosine`	COS	V	
`Zhang`	ZHANG	V and C_0	[30]

Table 3.14: Summary of operations in stage `INITIALIZE`.

Operation	Description
`UMeshImporterOp`	Loads unstructured surface mesh (UMesh)
`SurfaceComponentSetOp`	Creates a database for geometry components
`LogicalMeshSurfaceSugarcubeOp`	Performs sugarcubing to determine the relationship between UMesh and a structured volume mesh (SMesh)
`UFieldMSInitOp`	Creates a UMesh field object class
`MSPDistInitOp`	Creates a MSPDist object class
`MSPDistCombineOp`	Combines particle distributions - only needed for multiple sub-domain simulations
`MSPDistCellIDOp`	Determines SMesh cell IDs based on particle positions
`MSPDistSortOp`	Sorts particles in the order of cell ID (particles outside the domain are removed)
`MSPDistNormalMaxwellianStreamOp`	Injection of particles at a Maxwellian distribution
`MSPDistSourceRPAOp`	Injection of particles according to RPA data
`MSPDistSampleOp`	Samples MSPDist data to compute SMesh fields
`SampleFieldWriteVTKOp`	Writes sampling field data in VTK format
`MSPDistWriteVTKOp`	Writes MSPDist data in VTK format
`FieldSetOp`	Set a SMesh field to a value
`MSPDistChargeDepositionOp`	Deposits particle charges to a grid
`FieldArithmeticOp`	Divides by cell volumes to get charge densities
`LogicalFieldPatchOp`	Patches field data across sub-domains
`NextStageOp`	Goes to next stage

3.5.5 TURF Operations

Stage: Initialize

The INITIALIZE stage consists of operations which import objects and create parameters necessary for the simulation. The operations are listed and briefly described in Table 3.14.

The operation UMeshImporterOp imports the mesh file containing the spacecraft geometry. The spacecraft components can be made up of different materials and have parameters specific to those components as specified in an input file. The operation SurfaceComponentSetOp imports user-input parameters and values and creates a database. Then, the relationship between the surface mesh and the volume mesh is determined by LogicalMeshSurfaceSugarcubeOp. This operation can also initiate UMeshImporterOp and SurfaceComponentSetOp by defining these parameters within the block of LogicalMeshSurfaceSugarcubeOp.

```
DEFINE STAGE INITIALIZE
##########################################
## Load and Initialize Spacecraft Geometry ##
##########################################
DEFINE OPERATION
       TYPE = UMeshImporterOp
       UMESH_FILES = jaysat2.inp
       SCALE = 0.012
       TRANSFORM_CENTER = 0.0, 0.0, 0.0
       TRANSLATE = -0.15, 0.0, 0.0
       ROTATION_AXIS = 0.0, 0.0, 1.0
       ROTATION_DEGREES = 0.0
       UMESH_TYPE = Surface
       UMESH_FILE_TYPE = abaqus
       UMESH_NAME = UMesh
       WRITE_VTK_MESH = FALSE
END OPERATION
DEFINE OPERATION
       TYPE = SurfaceComponentSetOp
       SURFACE_NAME = UMesh
       DEFAULT_COMPONENT_MATERIAL_NAME = Al@s
       DEFAULT_COMPONENT_POTENTIAL = 0.0
       DEFAULT_COMPONENT_TEMPERATURE = 300.0
       COMPONENT_MATERIAL_FILE = component_TURF.txt
       VERBOSE = TRUE
END OPERATION
DEFINE OPERATION
       TYPE = LogicalMeshSurfaceSugarcubeOp
       SURFACE_NAME = UMesh
       SUGARCUBE_NAME = SC_UMesh
       SMESH_NAME = SMesh
       FIELD_NAME = bc
       EXTERIOR_PTS = 0.0, 0.0, 4.0
       WRITE_VTK_MESH = TRUE
END OPERATION
```

- SURFACE_FILE points to the file of the desired satellite geometry.
- SCALE allows the user to account for unit conversions if the geometry was not built in meters.
- The Transform, Translate, and Rotation commands determine the position and orientation of geometry.

Table 3.15: Surface mesh field available in TURF.

Variable Name	Description
NC	Number of simulation particle impacting a surface element, particle.
NPFLUX	Particle flux to a surface, particle/m^2/s.
MFLUX	Mass flux, kg/m^2/s.
PRES	Pressure, N/m^2.
TAU	Shear stress, N/m^2.
QFLUX	Heat flux, J/m^2/s.

- DEFAULT_COMPONENT_MATERIAL_NAME sets the default material for all parts of the geometry.[15]
- DEFAULT_COMPONENT_POTENTIAL sets the default potential for all parts of the geometry.
- COMPONENT_MATERIAL_FILE calls on the file containing information about the geometry components.[16]
- EXTERIOR_PTS are the x, y, and z coordinates of points exterior to the geometry. Multiple points can be defined as EXTERIOR_PTS = x1, y1, z1, x2, y2, z2, ...

A new unstructured mesh data field is created with UFieldMSInitOp.

```
############################
## Initialize UMesh Field ##
############################
DEFINE OPERATION
      TYPE = UFieldMSInitOp
      UFIELD_DATA_NAME = SurfaceField
      UMESH_NAME = UMesh
      FIELD_NAMES = NC NCIN NCOUT NPFLUX NPFLUX-IN NPFLUX-OUT MFLUX PRES TAUX TAUY TAUZ QFLUX
      SPECIES_NAMES = Xe+@g Xe@g Al@g Fe@g
END OPERATION
```

- SPECIES_NAME indicates particle species to be sampled and saved to the data fields

The MSPDist class objects that store particle information are created by MSPDistInitOp. P-DST is the primary particle distribution, P-GST and P-DEL are the temporary particle distributions, and P-EXC is the exchange particle distribution only used for particle patching between sub-domains.

```
################################################
## Particle and Ghost/Exchange Distributions ##
################################################
DEFINE OPERATION
      TYPE = MSPDistInitOp
      MSPDIST_DATA_NAME = P-DST
      MAX_NP = 1024000
      SPECIES_NAMES = Xe@g Xe+@g Al@g Fe@g
END OPERATION
DEFINE OPERATION
      TYPE = MSPDistInitOp
      MSPDIST_DATA_NAME = P-GST
      MAX_NP = 1024000
      SPECIES_NAMES = Xe@g Xe+@g Al@g Fe@g
```

[15] The value should be one of the materials named in the world file.
[16] If this command is used in addition to the default component initializations, values for components named in the material file have their default values overwritten.

```
END OPERATION
DEFINE OPERATION
       TYPE = MSPDistInitOp
       MSPDIST_DATA_NAME = P-DEL
       MAX_NP = 1024000
       SPECIES_NAMES = Xe@g Xe+@g Al@g Fe@g
END OPERATION
DEFINE OPERATION
       TYPE = MSPDistInitOp
       MSPDIST_DATA_NAME = P-EXC
       MAX_NP = 50000
       SPECIES_NAMES = Xe@g Xe+@g Al@g Fe@g
END OPERATION
```

- `MAX_NP` sets the maximum number particles to be used in the simulation.[17]
- `SPECIES_NAMES` loads gas species that will be available in the particle distribution.

Particles existing within ghost cells and stored in `P-EXC` are patched to an appropriate sub-domain from the previous stage. At this point, the particles still in ghost cells are essentially outside of the simulation domain, and the rest of particles are within non-ghost region of one of the sub-domains. `MSPDistCombineOp` combines `P-EXC` with the primary distribution `P-DST`. The combination of `MSPDistCellIDOp` and `MSPDistSortOp` removes them from the particle distribution.

```
#######################################################
## Combine Particles in Ghost Cells from Prior Step ##
#######################################################
DEFINE OPERATION
       TYPE = MSPDistCombineOp
       MSPDIST_SRC_NAME = P-EXC
       MSPDIST_DST_NAME = P-DST
       VERBOSE = FALSE
END OPERATION
# Update CellID
DEFINE OPERATION
       TYPE = MSPDistCellIDOp
       MSPDIST_DATA_NAME = P-DST
END OPERATION
# Sort removes particles with CellID>max
DEFINE OPERATION
       TYPE = MSPDistSortOp
       NAME = Sort_P-DST
       MSPDIST_SRC_NAME = P-DST
       MSPDIST_DST_NAME = P-GST
END OPERATION
```

Neutral atoms are injected from one-sided Maxwellian distribution, while ions are injected according to the current density distribution measured by the Retarded Potential Analyzer (RPA).

```
############################
## Injection of Particles ##
############################
```

[17] It is important to ensure not to exceed memory availability. If the particle count ever exceeds these values, TURF will crash with an error message

```
# Inject neutral atoms from surface
DEFINE OPERATION
       TYPE = MSPDistNormalMaxwellianStreamOp
       MSPDIST_DATA_NAME = P-DST
       TEMPERATURE = 24920.0
       NORMAL_VELOCITY = 5000.0
       MASS_FLOW_RATE = 1.0e-4
       REAL_TO_COMPUTATIONAL = 1.0e13
       SPECIES = Xe@g
       SURFACE_NAME = UMesh
       SURFACE_GROUP_NAME = HCT1_INJ
       VERBOSE = FALSE
END OPERATION
# Inject ions from surface
DEFINE OPERATION
       TYPE = MSPDistSourceRPAOp
       MSPDIST_DATA_NAME = P-DST
       SURFACE_NAME = UMesh
       COMPONENT_AT_SPHERE_CENTER = HCT1_INJ
       SPECIES = Xe+@g
       INPUT_CSV_FILE_NAME = RPA_HCT1.csv
       INJECTION_RADIUS = 0.3
       MASS_FLOW_RATE = 1.0e-4
       REAL_TO_COMPUTATIONAL = 1.0e13
       VERBOSE = FALSE
END OPERATION
```

- `TEMPERATURE` refers to the inflow temperature of the injected neutral atoms. Measured in Kelvin.
- `NORMAL_VELOCITY` is the streaming velocity with respect to injection surface in m/s.[18]
- `MASS_FLOW_RATE` dictates how many particles of each species are injected. Measured in kg/s.
- `REAL_TO_COMPUTATIONAL` is the ratio which allows the simulation to scale the number of particles to a manageable quantity as a realistic particle count would be too large for a reasonable simulation time.
- `SPECIES` determines what kinds of particles are injected by each operation. Only one species can be injected for each block of operation.
- `SURFACE_GROUP_NAME` establishes where the particles are injected. It should be the name given to the thruster face's block in the geometry file.
- `COMPONENT_AT_SPHERE_CENTER` places the RPA source at the center of the given component. It should be the name given to the thruster face's block in the geometry file.
- `INPUT_CSV_FILE_NAME` calls on the csv file containing the RPA data and imports the information to provide an injection source.
- `INJECTION_RADIUS` sets the size of the RPA injection sphere.

The operation `MSPDistSampleOp` samples particle count, velocity, and kinetic energy and converts them to the various field data defined in the World file.

```
##############################
## Sum to Fields for Output ##
##############################
# Sampling of neutral atom is done more frequently for elastic collision calculation
```

[18] Neutral atoms should nearly be thermal and are not likely to have a driven normal velocity, but the streaming velocity is set to a large value to accelerate the simulation in this example.

```
DEFINE OPERATION
       TYPE = MSPDistSampleOp
       FIELD_DATA_NAME = FieldData
       MSPDIST_DATA_NAME = P-DST
       FIELD_NAMES = n_Xe
       MIX_START_TIME = 2.0e-3
       STEPS_PER_SAMPLING = 1
       STEPS_PER_FIELD_CALCULATION = 1
END OPERATION
# Other fields only need to be computed when output
DEFINE OPERATION
       TYPE = MSPDistSampleOp
       NAME = Sample_P-DST
       FIELD_DATA_NAME = FieldData
       MSPDIST_DATA_NAME = P-DST
       FIELD_NAMES = VOL n n_Xe n_Xe+ n_Al n_Fe
       MIX_START_TIME = 2.0e-3
       STEPS_PER_SAMPLING = 1
       STEPS_PER_FIELD_CALCULATION = 100
END OPERATION
```

- `MIX_START_TIME` should be set to approximately the time when particle steady-state is reached. This is to improve the statistics by taking the time-average through multiple time-steps.
- `STEPS_PER_SAMPLING` is the number of iterations conducted before sample data is collected.
- `STEPS_PER_CALCULATION` is the number of iterations conducted before sample data is used for field calculations. Value should be equal to or a multiple of the operation's `STEPS_PER_SAMPLING` value.

`SampleFieldWriteVTKOp` writes the restart file for volume mesh sampling field, and `MSPDistWriteVTKOp` outputs particle data which can be used as a restart file.

```
########################
## Output for Restart ##
########################
DEFINE OPERATION
       TYPE = SampleFieldWriteVTKOp
       SAMPLE_OP_NAME = Sample_P-DST
       MSPDIST_DATA_NAME = P-DST
       FILE_HEAD = Restart/sampleA
       SKIP = 500
       FORMAT = Binary
       COMPRESS_DATA = TRUE
       LAST_ONLY = FALSE
END OPERATION
DEFINE OPERATION
       TYPE = MSPDistWriteVTKOp
       MSPDIST_DATA_NAME = P-DST
       SPECIES = ALL
       FILE_HEAD = Restart/particleA
       SKIP = 500
       PARTICLE_SKIP = 1.0
       FORMAT = Binary
       COMPRESS_DATA = TRUE
```

```
      OVERWRITE = TRUE
END OPERATION
```

- SKIP tells how frequently the restart file should be written.
- FORMAT is the format of written data. Both ASCII and Binary are supported.
- COMPRESS_DATA is set TRUE to compress the binary data with zlib routine.
- OVERWRITE is set TRUE if only one restart file to be kept.

These operations compute the charge density. When the first order weighting scheme is used for charge deposition, the charge density field needs to be patched across the sub-domains using LogicalFieldPatchOp.

```
#################################
## Deposit Charge for Boltzmann ##
#################################
DEFINE OPERATION
      TYPE = FieldSetOp
      FIELD_DATA_NAME = FieldData
      FIELD_NAME = rho
      VALUE = 0.0
      OPERATION = SET
END OPERATION
DEFINE OPERATION
      TYPE = MSPDistChargeDepositionOp
      FIELD_DATA_NAME = FieldData
      FIELD_NAME = rho
      MSPDIST_DATA_NAME = P-DST
      ORDER = 1
END OPERATION
# Divide by cell volume to get charge density
DEFINE OPERATION
      TYPE = FieldArithmeticOp
      FIELD_DATA_NAME = FieldData
      FIELD_DST_NAME = rho
      FIELD_SRC_NAME = rho
      OPERATION = DIV
      USE_VOLUME = TRUE
END OPERATION
#######################################
### Patch Rho before Boltzmann Solve ###
#######################################
DEFINE OPERATION
      TYPE = LogicalFieldPatchOp
      FIELD_NAME = rho
      PATCH_OP = PATCH_SUM
END OPERATION

###############################
## Proceed to the Next Stage ##
###############################
DEFINE OPERATION
      TYPE = NextStageOp
END OPERATION
```

```
END STAGE INITIALIZE

################################################################################
```

Stage: Field

The `FIELD` stage contains operations which unify the contiguous region across sub-domains, solve for electric potential via Boltzmann inversion, compute electric field, and output data. The list of operations used in this stage is shown in Table 3.16. The operation `LogicalFieldPatchOp` is called again to terminate patching after the `INITIALIZE` stage.

```
DEFINE STAGE FIELD

##########################################
### Stop Patching After INITIALIZE Stage ###
##########################################
DEFINE OPERATION
      TYPE = LogicalFieldPatchOp
      FIELD_NAMES = rho
      PATCH_OPS = PATCH_NULL
END OPERATION
```

The following operations add the lower bound for the charge density field, set the electric potential within the satellite geometries, and compute potential via Boltzmann inversion.

```
##########################################
## Potential Solve via Boltzmann Inversion ##
##########################################
# Adding background density for Boltzmann solve
DEFINE OPERATION
      TYPE = FieldArithmeticConstOp
      FIELD_DATA_NAME = FieldData
      FIELD_SRC_NAME = rho
      FIELD_DST_NAME = rho
      CONSTANT = 1.602e-7 # Minimum density
      OPERATION = Add
```

Table 3.16: Summary of operations in stage FIELD

Operation	Description
`LogicalFieldPatchOp`	Stops patching of field data
`FieldArithmeticConstOp`	Adds the minimum density such that the density is greater than zero everywhere in the domain for a Boltzmann potential solve
`FieldBlendTimeOp`	Time-averages charge density
`LogicalPotentialSetInsideGeometryOp`	Sets potential within a geometry
`LogicalPotentialBoltzmannOp`	Potential solve by a Boltzmann inversion
`LogicalNodeGradientOp`	Computes electric fields at nodes
`LogicalFieldWriteVTKOp`	Writes 3D data in VTK format
`LogicalFieldWriteVTK2DOp`	Writes 2D data in VTK format
`MSPDistWriteVTKOp`	Writes MSPDist data in VTK format
`MSPDistCopyOp`	Copies data in a MSPDist to other MSPDist
`NextStageOp`	Goes to next stage

```
      USE_GPU = False
END OPERATION
# Time average rho after TIME_START_AVERAGING
# rho_avg can be used for later calculations or just for plotting
DEFINE OPERATION
      TYPE = FieldBlendTimeOp
      FIELD_DATA_NAME = FieldData
      FIELD_SRC_NAME = rho
      FIELD_DST_NAME = rho_avg
      OPERATION = AVERAGE
      TIME_START_AVERAGING = 2.0e-3
END OPERATION
# Set potential inside geometries and flag poisson region
DEFINE OPERATION
      TYPE = LogicalPotentialSetInsideGeometryOp
      SUGARCUBE_NAME = SC_UMesh
      FIELD_DATA_NAME = FieldData
      FIELD_CHARGE_DENSITY_NAME = rho
      FIELD_POTENTIAL_NAME = phi
      POTENTIAL_MODEL = BOLTZMANN
      TEMPERATURE_MODEL = POLY # CONST
      REFERENCE_GAMMA = 1.3
      REFERENCE_ELECTRON_DENSITY = 3.11135E17
      REFERENCE_ELECTRON_TEMPERATURE = 8.01663
      REFERENCE_POTENTIAL = 35.7182
      ADD_FLAG_TO_FIELD = TRUE
      FIELD_POTENTIAL_FLAG_NAME = pflag
END OPERATION
# Compute potential using Boltzmann relations
DEFINE OPERATION
      TYPE = LogicalPotentialBoltzmannOp
      FIELD_DATA_NAME = FieldData
      FIELD_CHARGE_DENSITY_NAME = rho
      FIELD_POTENTIAL_NAME = phi
# These parameters can be defined here if not using LogicalPotentialSetInsideGeometryOp
# TEMPERATURE_MODEL = POLY # CONST
# REFERENCE_GAMMA = 1.3
# REFERENCE_ELECTRON_DENSITY = 3.11135E17
# REFERENCE_ELECTRON_TEMPERATURE = 8.01663
# REFERENCE_POTENTIAL = 35.7182
END OPERATION
```

- `TIME_START_AVERAGING` should be the same value as the `MIXING_START_TIME`, the approximate beginning of particle steady state.
- `REFERENCE_GAMMA` is the effective polytropic specific heat ratio.
- `REFERENCE_ELECTRON_DENSITY` is the reference electron density in Boltzmann relations in m^{-3}.
- `REFERENCE_ELECTRON_TEMPERATURE` is the reference electron temperature in Boltzmann relations in eV.
- `REFERENCE_POTENTIAL` is the reference potential in Boltzmann relations in V.

Based on the electric potential at the cell-center locations, the electric field is computed at the nodes by simply applying the finite difference equation.

```
############################
## Compute Electric Field ##
############################
DEFINE OPERATION
       TYPE = LogicalNodeGradientOp
       FIELD_DATA_NAME = FieldData
       FIELD_POTENTIAL_NAME = phi
       FIELD_GRADIENT_PREFIX = En
       FIELD_MULTIPLY_CONSTANT = -1.0
       FIELD_GRADIENT_DIRECTIONS = [x, y, z]
END OPERATION
```

The initial data for all the fields will be saved before the simulation begins to move particles. The particle data are overwritten every 100 iterations, which can be used to restart the simulation from the particle state that TURF is terminated.

```
#########################################################################
## Write VTK Output - Moved Pre-Push for Consistent Patched Particles ##
#########################################################################
# 3D data
DEFINE OPERATION
       TYPE = LogicalFieldWriteVTKOp
       FIELD_DATA_NAME = FieldData
       FILE_HEAD = Plot3D/plt_
       FIELD_NAMES = VOL bc pflag rho rho_avg phi Enx Eny Enz n n_Xe n_Xe+ n_Al n_Fe
       SKIP = 100
       HELP = TRUE
       PLOT_GHOST = FALSE
END OPERATION
# Particle trajectories
# *** To view in Paraview, use "Temporal Particles To Pathline" filter,
# *** and change "Id Channel Array" to index
DEFINE OPERATION
       TYPE = MSPDistWriteVTKOp
       MSPDIST_DATA_NAME = P-DST
       SPECIES = ALL
       FILE_HEAD = PlotTraj/plt_
       SKIP = 100
       PARTICLE_SKIP = 1.0
       OVERWRITE = FALSE
END OPERATION
```

- Data for the fields specified in `FIELD_NAMES` will be exported into a folder named by `FILE_HEAD`.

These operations prepare the particles for movement before proceeding with the simulation and beginning the `MOVE` stage. `MSPDistCopyOp` copies particles stored to `P-DST` to `P-GST` and is required before performing the particle-surface interaction calculation in TURF. The last operation in this stage again directs TURF to proceed to the next stage.

```
########################################
## Prep for Advance Particle Positions ##
```

```
########################################
# Copy the distribution from P-DST to P-GST (P-DST is initialized)
DEFINE OPERATION
       TYPE = MSPDistCopyOp
       MSPDIST_SRC_NAME = P-DST
       MSPDIST_DST_NAME = P-GST
END OPERATION

##############################
## Proceed to the Next Stage ##
##############################
DEFINE OPERATION
       TYPE = NextStageOp
END OPERATION

END STAGE FIELD

################################################################################
```

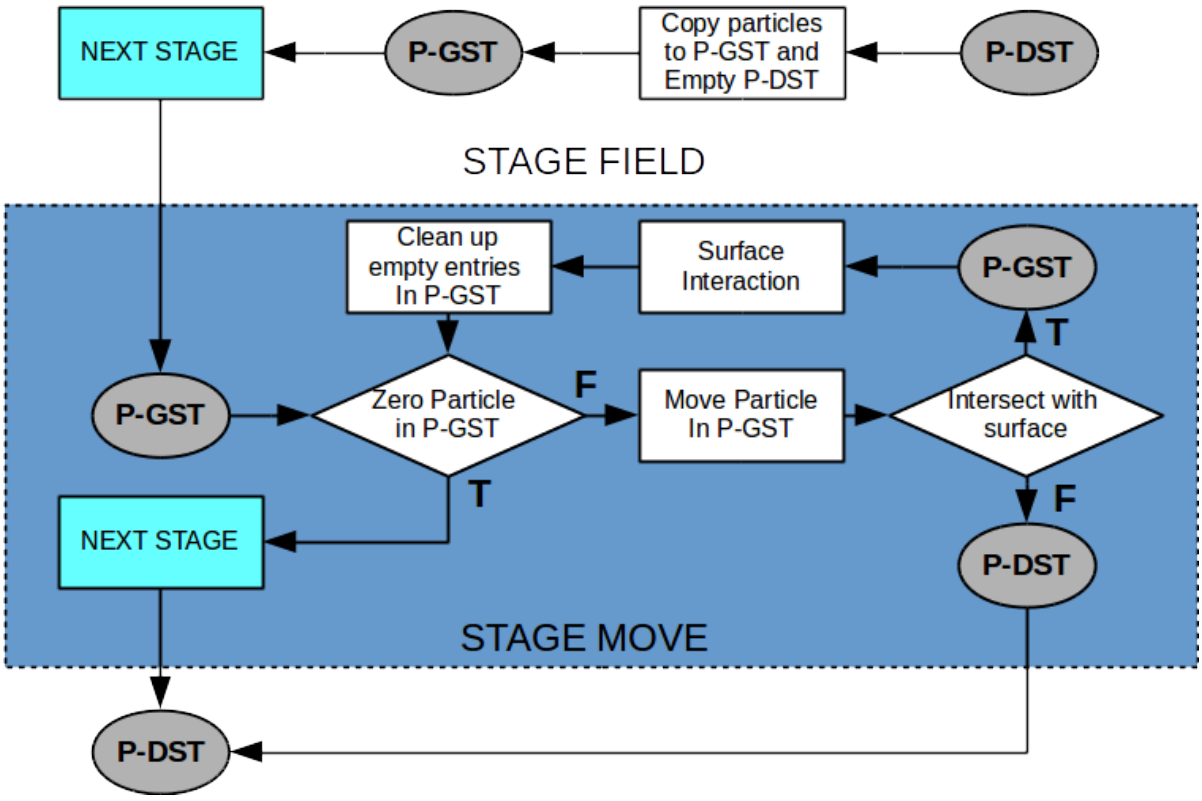


Fig. 3.11: Flowchart of stage `MOVE`

Stage: Move

The `MOVE` stage has operations which advance particles in an electric field and perform surface interaction calculations. This stage also performs calculations for numerical probes in order to compare the result with experimental

data. The list of operations in MOVE stage is given in Table 3.17. Furthermore, the general flow of the surface interaction calculation in TURF is illustrated in Fig. 3.11.

The first operation of stage MOVE pushes particles using the electric field stored at the nodes.

```
DEFINE STAGE MOVE
######################################################################
## Keep Advancing Particles Until Reaching the Final Time Per Step ##
######################################################################
# Advance P-GST for a full time-step
DEFINE OPERATION
      TYPE = MSPDistESPushOp
      FIELD_DATA_NAME = FieldData
      FIELD_EN_PREFIX = En
      FIELD_EN_DIRECTIONS = [x, y, z]
      MSPDIST_DATA_NAME = P-GST
      SPECIES_NAMES = Xe@g Xe+@g Al@g Fe@g
      USE_CFL_TIME_STEP = FALSE
      CFL_NUMBER = 0.5
END OPERATION
```

- SPECIES_NAMES specifies which particles are to be moved with the particle distribution during each time step.

TURF is able to mimic Faraday and RPA probes to measure current densities and energy distributions. These pseudo probes can be placed at any component of the spacecraft or along a virtual spherical surface and told to begin collecting data at any point in time in the simulation.

Probes using MSPDistProbeStageSphericalOp operations take measurements across a virtual sphere, and both the Faraday and RPA type probes can be used.

```
# Probe sweeping
# *** This needs to be done before MSPDistSugarcubeSurfIntersectionOp
# *** since particles in P-GST are marked for deletion by setting the
# *** weight to zero in that routine, but the weight is needed for
# *** probe measurement
DEFINE OPERATION
      TYPE = MSPDistProbeStageSphericalOp
      NAME = FARADAY_Axi
      MSPDIST_DATA_NAME = P-GST
      PROBE_TYPE = FARADAY # FARADAY or RPA
```

Table 3.17: Summary of operations in stage MOVE

Operation	Description
MSPDistESPushOp	Advances particles using node-centered electric fields
MSPDistProbeStageSphericalOp	Spherical stage probe
MSPDistSugarcubeSurfIntersectionOp	Determines if particles intersect with surface mesh during a time-step
MSPDistProbeFixedOp	Probe fixed to a surface
MSPDistSurfaceInteractionOp	Performs sputtering and surface reflections
MSPDistRemovePartOp	Removes particles that stuck to a surface or reached the final time within the stage
MSPDistParticleCountOp	Particle count
CriteriaStageOp	Repeats this stage until particle count becomes zero

```
      SURFACE_NAME = UMesh
      COMPONENT_AT_SPHERE_CENTER = HCT1_INJ
      SWEEP_RADIUS = 0.1
      PROBE_POLAR_ANGLE = 1.0
      SWEEP_POLAR_ANGLE_START = 0.0
      SWEEP_POLAR_ANGLE_END = 120.0
      NUMBER_OF_POLAR_ANGLE_BINS = 120
      VIEW_ANGLE = 30.0 # in degrees
      START_TIME = 2.0e-3
      STEPS_PER_SAMPLING = 1 # Samples every (SAMPLING) iteration
      VERBOSE = FALSE
END OPERATION
DEFINE OPERATION
      TYPE = MSPDistProbeStageSphericalOp
      NAME = RPA_Axi
      MSPDIST_DATA_NAME = P-GST
      PROBE_TYPE = RPA # FARADAY or RPA
      SURFACE_NAME = UMesh
      COMPONENT_AT_SPHERE_CENTER = HCT1_INJ
      SWEEP_RADIUS = 0.1
      PROBE_POLAR_ANGLE = 10.0
      SWEEP_POLAR_ANGLE_START = 0.0
      SWEEP_POLAR_ANGLE_END = 120.0
      NUMBER_OF_POLAR_ANGLE_BINS = 12
      MINIMUM_ENERGY = -0.5 # in eV - only for RPA
      MAXIMUM_ENERGY = 500.5 # in eV - only for RPA
      NUMBER_OF_ENERGY_BINS = 501 # - only for RPA
      VIEW_ANGLE = 15.0 # in degrees
      START_TIME = 2.0e-3
      STEPS_PER_SAMPLING = 1 # Samples every (SAMPLING) iteration
      VERBOSE = FALSE
END OPERATION
```

- COMPONENT_AT_SPHERE_CENTER specifies the geometry component that the measurement sphere is referenced to. The center of the spherical surface collecting currents is placed at the center of component specified here.
- SWEEP_RADIUS specifies the radius of the measurement sphere.
- SWEEP_PROBE _ANGLE_START and SWEEP_PROBE _ANGLE_END are the first and last polar angles of the measurement sphere to sweep the probe.
- NUMBER_OF_BINS divides the range of angles into the desired number of equally-sized sections. Measurements are taken at the center of each bin, and the thickness of each bin is determined by PROBE_POLAR_ANGLE.
- Current is only collected if the incident angle is smaller than the VIEW_ANGLE.
- START_TIME specifies the time to start the probe measurement.
- Measurements of current are only collected at multiples of the STEPS_PER_SAMPLING value.
- MAXIMUM_ENERGY and MINIMUM_ENERGY set the energy detection limits (in eV) for the RPA probe.
- NUMBER_OF_ENERGY_BINS divides the range of energies into the desired number of equally-sized sections.

The operation MSPDistSugarcubeSurfIntersectionOp determines whether particles impact surface mesh elements and moves these particle data to P-DST. The list of particle impacting elements is stored here and used when performing the surface reflection and/or sputtering calculations.

```
# Find intersection of particle trajectories with UMesh
# *** Starting from P-GST, particles are copied to P-DST once reaching the final time
# *** Then, P-GST contains particles bouncing at surface during the time-step
```

```
# *** And, P-DST contains particles making it to the final time per iteration
DEFINE OPERATION
       TYPE = MSPDistSugarcubeSurfIntersectionOp
       NAME = Intersection_P-GST
       MSPDIST_SRC_NAME = P-GST
       MSPDIST_DST_NAME = P-DST
       MSPDist_ERROR_NAME = P-EXC
       SUGARCUBE_NAME = SC_UMesh
       SURFACE_NAME = UMesh
       SURFACE_TEST = FALSE
       VERBOSE = FALSE
END OPERATION
```

The other kind of probe operations are `MSPDistProbeFixedOp` operations, which attach an unmoving probe to a component of the spacecraft surface mesh. The surface component can be specified as a virtual surface by specifying the material as "virtual" (`material:virtual`) in `component_TURF.txt`, and using this surface for this operation. In this way, particles can fly through this surface while probe measurement is taken.

```
# Fixed probe
DEFINE OPERATION
       TYPE = MSPDistProbeFixedOp
       NAME = RPA_Fixed
       URFACE_NAME = UMesh
       MSPDIST_DATA_NAME = P-GST
       SURF_INTERSECTION_OP_NAME = Intersection_P-GST
       COMPONENT_NAMES = SP_E1
       PROBE_TYPE = RPA # FARADAY or RPA
       VIEW_ANGLE = 90.0 # in degrees
       MINIMUM_ENERGY = -0.5 # in eV - only for RPA
       MAXIMUM_ENERGY = 500.5 # in eV - only for RPA
       NUMBER_OF_ENERGY_BINS = 501 # - only for RPA
       START_TIME = 2.0e-3
       STEPS_PER_SAMPLING = 1 # Samples every (SAMPLING) iteration
       VERBOSE = FALSE
END OPERATION
```

- `COMPONENT_NAMES` dictates which spacecraft component the probe is attached to.
- Current is only collected if the incident angle is smaller than the `VIEW_ANGLE`.
- `MAXIMUM_ENERGY` and `MINIMUM_ENERGY` set the energy detection limits (in eV) for the RPA probe.

The operation `MSPDistSugarcubeSurfInteractionOp` performs surface reflection and sputtering for different combinations of source and target species defined in `surf_int.txt`.

```
# Surface interaction - Reflection, sticking, and sputtering
DEFINE OPERATION
       TYPE = MSPDistSurfaceInteractionOp
       NAME = Interaction_P-GST
       SURFACE_NAME = UMesh
       MSPDIST_DATA_NAME = P-GST
       UFIELD_DATA_NAME = SurfaceField
       SURF_INTERSECTION_OP_NAME = Intersection_P-GST
       FILE_NAME = surf_int.txt
       STEPS_PER_SAMPLING = 1
```

```
      VERBOSE = FALSE
END OPERATION
```

While MSPDistSugarcubeSurfIntersectionOp moves particles reaching the end time within each time-step to P-GST, these particles still need to be deleted from P-DST. This is done by marking them in P-DST by setting their weights to be zero and actually removing them from P-DST by calling MSPDistRemovePartOp.

```
# Remove particles with zero weight
DEFINE OPERATION
      TYPE = MSPDistRemovePartOp
      MSPDIST_SRC_NAME = P-GST
      MSPDIST_DST_NAME = P-DEL
      REMOVE_ZERO_WEIGHT = TRUE
      REMOVE_CELLID_GT_MAX = FALSE
      DISCARD_REMOVED_PARTICLE = TRUE
END OPERATION
```

The following operations monitor the particle count in P-GST and proceed to the next stage if the particle count becomes zero, otherwise this stage is repeated.

```
#######################################################
## Quantity Needs To Be Smaller Than CRITERIA_LONG ##
#######################################################
# Obtain particle count in P-GST
DEFINE OPERATION
      TYPE = MSPDistParticleCountOp
      MSPDIST_DATA_NAME = P-GST
      QUANTITY_NAME = MSPDistCount
      VERBOSE = FALSE
END OPERATION

#############################################################
## Proceed to the Next Stage If Criterion is Satisfied ##
#############################################################
# Go to next stage if P-GST->Nparts is smaller than 1
DEFINE OPERATION
      TYPE = CriteriaStageOp
      QUANTITY_NAME = MSPDistCount
      CRITERIA_LONG = 1 #Go to next stage if P-GST->Nparts is smaller than 1
      MAX_ITERATION = 100 #Max number of subcycles to prevent stuck particles
END OPERATION

END STAGE MOVE

#############################################################################
```

Stage: Postmove

The stage POSTMOVE contains operations that implement effects of elastic collisions between particles, initiate the particle patching across sub-domains, compute surface field data, and send calculation data to various output files.

The first operation in this stage is MSPDistMCCElasticFitOp, which performs the elastic collision calculations between high energy ions and the background neutral atoms via Monte Carlo Collision model. Here, the elastic collision calculations are split into two mechanisms, i.e. momentum exchange (MEX) and charge exchange (CEX) interactions, and each interaction is approximated independently by MSPDistMCCElasticFitOp.

```
DEFINE STAGE POSTMOVE

############################
## Collision Calculations ##
############################
DEFINE OPERATION
       TYPE = MSPDistMCCElasticFitOp
       MSPDIST_DATA_NAME = P-DST
       SPECIES = Xe+@g
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = n_Xe
       TEMPERATURE_OF_TARGET_SPECIES = 300.0
       MEX_OR_CEX = MEX
       SIGMA_COEFFS = -27.2 171.13 1.0e-20
       FIRST_FIT_EXPONENT_DIFF_SIGMA = -2.02
       SECOND_FIT_EXPONENT_DIFF_SIGMA = 3.24
       FIT_ANGLE_DIFF_SIGMA = 3.526E-5
END OPERATION
DEFINE OPERATION
       TYPE = MSPDistMCCElasticFitOp
       MSPDIST_DATA_NAME = P-DST
       SPECIES = Xe+@g
       FIELD_DATA_NAME = FieldData
       FIELD_NAME = n_Xe
       TEMPERATURE_OF_TARGET_SPECIES = 300.0
       MEX_OR_CEX = CEX
       SIGMA_COEFFS = -27.2 171.13 1.0e-20
       FIRST_FIT_EXPONENT_DIFF_SIGMA = -1.098
       SECOND_FIT_EXPONENT_DIFF_SIGMA = 1.53
       FIT_ANGLE_DIFF_SIGMA = 1.375E-3
END OPERATION
```

- `SPECIES` is the species of the incident particle.
- `FIELD_NAME` dictates the density of the target gas.
- `SIGMA_COEFFS` represent the coefficients of the CEX collision cross-section, c_0, c_1, and c_2 in Eq. (3.10).
- `FIRST_FIT_EXPONENT_DIFF_SIGMA` is the fitting parameter for differential cross-section, A in Eq. (3.11).

Table 3.18: Summary of operations in stage POSTMOVE

Operation	Description
`MSPDistMCCElasticFitOp`	Performs MCC elastic collisions
`MSPDistSplitOp`	Splits MSPDist class object into two. Particles in ghost cells are copied into P-EXC that are later moved to P-DST in the other sub-domain
`GSOPatchOp`	Patches MSPDist across sub-domains
`UFieldMSComputeOp`	Computes surface mesh field
`UFieldWriteVTKOp`	Writes surface mesh field in VTK format
`ProbeStageWriteOp`	Writes spherical stage probe data to a csv file
`ProbeFixedWriteOp`	Writes fixed probe data to a csv file
`NextStageOp`	Goes to next stage

- `SECOND_FIT_EXPONENT_DIFF_SIGMA` is the fitting parameter for differential cross-section, B in Eq. (3.11).
- `FIT_ANGLE_DIFF_SIGMA` is the cut-off angle for differential cross-section.

In this model, the CEX collision cross-section is typically used for the MEX collision cross-section, so the coefficients that feed into `MSPDistMCCElasticFitOp` (`SIGMA_COEFF`) are the same. This is a fair assumption since the CEX cross-section is sufficiently large to capture all the impact parameters that yield large angle scattering. The equation for the total cross-section is solely dependent on the relative velocity v_{rel} and requires three independent parameters.

$$\sigma = \{c_0 \log(v_{\mathrm{rel}}) + c_1\}\, c_2 \tag{3.10}$$

In using this model, the differential cross-section for a specific combination of ion and atom at a given energy has to be available in literature. The differential cross-section can also be theoretically derived if the interaction potential between the incident and target species is available. Once obtaining the differential cross-section, fitting parameters are determined according to the following equation.

$$\left.\frac{d\sigma}{d\Omega}\right|_{\mathrm{LAB}} = \theta^{A_{\mathrm{MEX}}} 10^{B_{\mathrm{MEX}}} + (90-\theta)^{A_{\mathrm{CEX}}} 10^{B_{\mathrm{CEX}}} \tag{3.11}$$

Here, when an angle is smaller than the cut-off value, it is assumed that the differential cross-section is constant beyond the angle for the MEX collision. For the CEX collision, the same assumption is made when an angle is greater than the cut-off value. The cut-off angles are determined such that the integrated differential cross-section matches with the experimentally determined total cross-section. For more details of this model and implementation, readers should refer to Ref. [31].

As covered briefly in the `INITIALIZE` stage, particle patching is performed across two stages, and this stage is the first part of the particle patching. In an ideal simulation, particles can only move to, at most, the neighboring volume mesh cell. Therefore, particles can only be within a sub-domain or ghost cell region and cannot jump to outside of the ghost cell region. These particles are split into a separate particle distribution by `MSPDistSplitOp`. When a particle is in the ghost cell region, this particle can either be within the next sub-domain or outside the whole simulation domain. The next operation `GSOPatchOp` passes the particles to the corresponding sub-domain or keep it in the current sub-domain if they do not belong to any. Then, in the `INITIALIZE` stage, particles outside of simulation domain are deleted.

```
############################
## Perform Particle Patch ##
############################
# Particle CellID greater than maximum is copied to P-EXC
DEFINE OPERATION
      TYPE = MSPDistSplitOp
      MSPDIST_SRC_NAME = P-DST
      MSPDIST_DST_NAME = P-EXC
END OPERATION
# Patch particles outside sub-domains
DEFINE OPERATION
      TYPE = GSOPatchOp
      SRC_NAME = P-EXC
      DST_NAME = P-EXC
END OPERATION
```

These operations compute and output the surface mesh field though the data sampled in `MSPDistSurface-InteractionOp`.

```
############################
## Sum to Fields for Output ##
############################
# Compute UField quantities
```

```
# *** This needs to be after surface interaction
DEFINE OPERATION
       TYPE = UFieldMSComputeOp
       UFIELD_DATA_NAME = SurfaceField
       SURF_INTERACTION_OP_NAME = Interaction_P-GST
       MIX_START_TIME = 2.0e-3
       STEPS_PER_FIELD_CALCULATION = 100
END OPERATION
DEFINE OPERATION
       TYPE = UFieldWriteVTKOp
       UFIELD_DATA_NAME = SurfaceField
       FILE_HEAD = PlotSurf/pltSurf_
       FIELD_NAME = NPFLUX-IN_Xe@g NPFLUX-IN_Xe+@g NPFLUX-IN_Fe@g NPFLUX-OUT_Al@g NPFLUX-OUT_Fe@g
       SKIP = 100
END OPERATION
```

- MIX_START_TIME is the time to start averaging between iterations for the surface fields in order to improve statistics.
- FIELD_NAME specifies which fields are exported into the FILE_HEAD folder.
- SKIP tells TURF how many iterations to wait/skip before the field information is exported. This value should be equal to (or a multiple of) the STEPS_PER_CALCULATION value in the UFieldMSComputeOp operation preceding it.

The following operations export the data collected by the pseudo probes from the MOVE stage to csv files named in FILE_HEAD.

```
########################
## Output probe data ##
########################
DEFINE OPERATION
       TYPE = ProbeStageWriteOp
       PROBE_OP_NAME = FARADAY_Axi
       STEPS_PER_OUTPUT = 100 # Averages out OUTPUT/SAMPLING data
       TIME_DEPENDENT_DATA = FALSE # Overwrites the same file if false
       RESET_SAMPLING = FALSE
       FILE_HEAD = Probe/FARADAY_Axi
END OPERATION
DEFINE OPERATION
       TYPE = ProbeStageWriteOp
       PROBE_OP_NAME = RPA_Axi
       STEPS_PER_OUTPUT = 100 # Averages out OUTPUT/SAMPLING data
       TIME_DEPENDENT_DATA = FALSE # Overwrites the same file if false
       RESET_SAMPLING = FALSE
       FILE_HEAD = Probe/RPA_Axi
END OPERATION
DEFINE OPERATION
       TYPE = ProbeFixedWriteOp
       PROBE_OP_NAME = RPA_Fixed
       STEPS_PER_OUTPUT = 100 # Averages out OUTPUT/SAMPLING data
       TIME_DEPENDENT_DATA = FALSE # Overwrites the same file if false
       RESET_SAMPLING = FALSE
       FILE_HEAD = Probe/RPA_Fixed
```

```
END OPERATION
```

- Current measurements are collected very frequently in the `MOVE` stage. A larger `STEPS_PER_OUTPUT` value smooths out the data.

Before the end of iteration, the particle count is accumulated across sub-domains to obtain the global values (total as well as per species). This information is useful when determining whether the simulation has reached steady-state. The particle count information can be output on the terminal by enabling `VERBOSE` option and can be written to a file by specifying `LOG_FILE`.

```
#######################################
## Count Particles Prior to Injection ##
#######################################
DEFINE OPERATION
      TYPE = MSPDistParticleCountOp
      MSPDIST_DATA_NAME = P-DST
      QUANTITY_NAME = ParticleCount
      LOG_FILE = ParticleCount.csv
      VERBOSE = TRUE
      GLOBAL = TRUE
END OPERATION

##############################
## Proceed to the Next Stage ##
##############################
DEFINE OPERATION
      TYPE = NextStageOp
END OPERATION

END STAGE POSTMOVE

#############################################################################
```

3.5.6 Restart

In TURF-IR version 2017a, TURF simulation can only be restarted through particle data output by `MSPDistWriteVTKOp` and sampling field data output by `SampleFieldWriteVTKOp`. Both of these operations are used in the electric propulsion plume example demonstrated in Section 3.5.5. While the simulation in Section 3.5.5 is run, restart files are output in the `Restart` folder. This section demonstrates how to perform a restart from these restart files. In order to run a simulation with restart, we will update the symbolic link to `world.list`.

```
      tutorial-TURF/TURF-IR_v017a/Plume/PlumeExample> ln -sf world.restart.list world.list
```

The World file, `world.restart.list`, is essentially the same as `world.sat.list` except for the simulation end time and operations file. The operations file used in this simulation is `operations.restart.list`, which has two additional operations, `MSPDistReadVTKOp` and `SampleFieldReadVTKOp`. `MSPDistReadVTKOp` reads the particle data stored in MSPDist class object, and `SampleFieldReadVTKOp` reads the sampling field data.

```
###########################################
## Extract Particle Data from Restart File ##
###########################################
DEFINE OPERATION
      TYPE = MSPDistReadVTKOp
```

```
      MSPDIST_DATA_NAME = P-DST
      FILE_HEAD_PVTP = Restart/particleA
      UPDATE_WORLD_TIME = TRUE
      ADVANCE_TIME_BY_DT = TRUE
      REINDEX = FALSE
END OPERATION
```

```
################################
## Restart Sampled Field Data ##
################################
DEFINE OPERATION
      TYPE = SampleFieldReadVTKOp
      SAMPLE_OP_NAME = Sample_P-DST
      MSPDIST_DATA_NAME = P-DST
      FILE_HEAD = Restart/sampleA
      UPDATE_WORLD_TIME = FALSE #Taken care of by MSPDistReadVTKOp
      ADVANCE_TIME_BY_DT = FALSE
      RESET_SAMPLING = FALSE
END OPERATION
```

- `UPDATE_WORLD_TIME` is set TRUE if the simulation time is updated with the time specified in the restart file.
- `ADVANCE_TIME_BY_DT` is set TRUE if the simulation time needs to be advanced by one time-step. Whether to advance the time depends on which stage the restart files are written.
- `REINDEX` is set TRUE if particle indices are to be redefined. Particle index is redefined by the domains that a particle resides, but this will be meaningless if number of domains is changed after the restart.
- `RESET_SAMPLING` is set TRUE if sampling is reset.

In order to perform a restart for sampling field, the domains have to be set up exactly in the same way between the restarts. However, the particle restart does not have such a requirement that the simulation domain can be partitioned differently and grid resolution can be changed between restarts.

3.5.7 Total Yield

`Constant`

This is the simplest among all the sputter models implemented in TURF. The total yield Y is simply a constant value specified in the input.

$$Y = C_0$$

`Matsunami`

Although we call this model "Matsunami", the original work was actually published by Yamamura, Matsunami, and Itoh [22]. Only two inputs (C_0 and C_1) are required for the model. When initializing TURF, four constants (A_0, A_1, A_2, and A_3) are precomputed based off the inputs and material properties.

$$\begin{aligned}
A_0 &= \frac{0.03255}{Z_1 Z_2 (Z_1^{2/3} + Z_2^{2/3})^{1/2}} \frac{M_2}{M_1 + M_2} \\
A_1 &= 0.079 \frac{(M_1 + M_2)^{2/3}}{M_1^{3/2} M_2^{1/2}} \frac{Z_1^{2/3} Z_2^{1/2}}{(Z_1^{2/3} + Z_2^{2/3})^{3/4}} \\
A_2 &= U_s \left(1.9 + 3.8(M_2/M_1)^{-1} + 0.134(M_2/M_1)^{1.24}\right) \\
A_3 &= \alpha^* Q K
\end{aligned}$$

where

$$\alpha^* = 0.08 + 0.164(M_2/M_1)^{0.4} + 0.0145(M_2/M_1)^{1.29}$$
$$K = 8.478 \frac{Z_1 Z_2}{(Z_1^{2/3} + Z_2^{2/3})^{1/2}} \frac{M_1}{M_1 + M_2}$$

Here U_s and Q are used in place of C_0 and C_1, respectively, in order to keep the symbols used in the original reference. These parameters can be found in Ref. [22]. Z is the atomic number in amu and M is the atomic mass in amu. Subscripts 1 and 2 denote source and target particles. Then, the total yield is computed for each sputtering event by

$$Y = 0.42 \frac{A_3 s_n}{U_s(1 + 0.35 U_s s_e)} \left[1 - \sqrt{\frac{A_2}{E}}\right]^{2.8}$$

where s_n and s_e are

$$s_n = \frac{3.441\sqrt{\epsilon}\ln(\epsilon + 2.718)}{1 + 6.355\sqrt{\epsilon} + \epsilon(-1.708 + 6.882\sqrt{\epsilon})}$$
$$s_e = A_1\sqrt{\epsilon}$$

where $\epsilon = A_0 E$ and E is the energy of incident particle in electronvolt.

`Yamamura`

Yamamura and Tawara [24] upgraded the model originally developed by Yamamura, Matsunami, and Itoh (referred as `Matsunami` model herein). Six inputs (C_0 to C_5) are required for this model, and seven constants are stored during the initialization stage.[19]

$$B_0 = 0.42\alpha^* Q K / U_s$$
$$B_1 = \frac{W}{1 + (M_1/7)^3} 0.079 \frac{(M_1 + M_2)^{3/2}}{M_1^{3/2} M_2^{1/2}} \frac{Z_1^{2/3} Z_2^{1/2}}{(Z_1^{2/3} + Z_2^{2/3})^{3/4}}$$
$$B_2 = E_{th}$$
$$B_3 = \frac{0.03255}{Z_1 Z_2 (Z_1^{2/3} + Z_2^{2/3})^{1/2}} \frac{M_2}{M_1 + M_2}$$
$$B_4 = s$$
$$B_5 = E_{th,\theta} = 1.5 \frac{U_s}{\gamma} \left[1 + 1.38 \left(\frac{M_1}{M_2}\right)^h\right]^2$$
$$B_6 = (a/R_0)^{1.5} \sqrt{Z_1 Z_2 / (Z_1^{2/3} + Z_2^{2/3})^{1/2}}$$
$$B_7 = f_s$$

where

[19] Yamamura's paper has typo in his equation where the coefficient should be 0.042 in the equation for $Y(E)$

$$\alpha^* = \begin{cases} 0.249(M_2/M_1)^{0.56} + 0.0035(M_2/M_1)^{1.5} & M_1 \le M_2 \\ 0.0875(M_2/M_1)^{-0.15} + 0.165(M2/M1) & \text{Otherwise} \end{cases}$$

$$h = \begin{cases} 0.834 & M_1 \le M_2 \\ 0.18 & \text{Otherwise} \end{cases}$$

$$K = 8.478 \frac{Z_1 Z_2}{(Z_1^{2/3} + Z_2^{2/3})^{1/2}} \frac{M_1}{M_1 + M_2}$$

$$\gamma = \frac{4 M_1 M_2}{(M_1 + M_2)^2}$$

$$a = \begin{cases} \dfrac{0.4685}{(Z_1^{1/2} + Z_2^{1/2})^{2/3}} & Z_1 > 3 \\ \dfrac{0.4685}{(Z_1^{2/3} + Z_2^{2/3})^{1/2}} & \text{Otherwise} \end{cases}$$

$$E_{th} = \begin{cases} U_s \frac{1.0 + 5.7 M_1/M_2}{\gamma} & M_1 \le M_2 \\ U_s \frac{6.7}{\gamma} & \text{Otherwise} \end{cases}$$

Here U_s, Q, W, s, f_s, R_0 are used in place of C_0, C_1, C_2, C_3, C_4, and C_5, respectively. The input parameter for various combinations of source and target materials can be found in Table 4 and 17 of Ref. [22] and Table 1 of Ref. [24]. Then, the yield for every incident particle is computed by

$$Y(E) = \beta \frac{B_0 s_n}{1 + B_1 \epsilon^{0.3}} \left(1 - \sqrt{\frac{B_2}{E}}\right)^{B_4}$$

where

$$\epsilon = B_3 E, \quad s_n = \frac{3.441\sqrt{\epsilon}\log(\epsilon + 2.718)}{1.0 + 6.355\sqrt{\epsilon} + \epsilon(-1.708 + 6.882\sqrt{\epsilon})}$$

$$\nu = 1 - \sqrt{B_5/E}, \quad f = B_7 \frac{1 + 2.5(1 - \nu)}{\nu}$$

$$\sigma = f\cos(\theta_{\text{opt}}), \quad \theta_{\text{opt}} = 0.5\pi - 4.9916(B_6/\sqrt{E})^{0.45}$$

$$\beta = \frac{\exp(-\sigma(1/\cos\theta_i - 1))}{\cos^f \theta_i}$$

Here, E is the incident particle energy in electronvolt.

`Kannenberg`

The `Kannenberg` sputter yield [26] is modeled as a function of energy, E, and incident angle, β. The total yield in atoms/ion is given as follows.

$$Y(E, \beta) = S(E) g(\beta)$$
$$S(E) = C_0 + C_1 E$$
$$g(\beta) = C_2 \cos\theta + C_3 \cos^2\theta + C_4 \cos^3\theta$$

If the energy threshold($E_{th} = -a/b$) is negative, the fit is rejected on physical grounds and is instead constrained to be zero. The reference paper shows a lot of data for ITO but has no coefficients in it at all. Assume that all AFRL `Kannenberg` data fits are Lockheed proprietary.

`Roussel`

The sputtering yield of standard coverglass (cerium-doped borosilicate) as a function of incidence angle, β, was measured and fit to a polynomial (for energy 300 eV) [27]. This resulted in the following yield Where β is in degrees and Y is in atoms/ion:

$$Y(E = 300eV, \beta) = 0.42 - 0.0053\beta + 0.0015\beta^2 - 7 \times 10^{-6}\beta^3 - 10^{-7}\beta^4$$

The coverglass had a normal yield of 0.42 for 300 eV impact ions and a linear dependence from a threshold energy of 50 eV. This allows for separation of the yield into

$$Y(E,\beta) = S(E)g(\beta)$$

where $S(E) = -0.084 + 0.0017E$. This form is generalized as follows.

$$\begin{aligned} Y(E,\beta) &= S(E)g(\beta) \\ S(E) &= C_0 + C_1 E \\ g(\beta) &= 1 - 0.723\beta + 11.724\beta^2 - 3.13\beta^3 - 2.57\beta^4 \end{aligned}$$

where β is redefined to be in radians. Unlike the `Kannenberg` sputter model, $g(\beta)$ is a polynomial function in β instead of $\cos\beta$. Therefore, the set of coefficients may not yield a sputter yield of zero at $\beta = 90°$.

`Garnier`

The `Garnier` sputtering algorithm performs yield calculations according to Garnier, et al for Boron Nitride [28]. `Garnier` yield fits are simplified equations from `Yamamura` which have a separate angle dependence and energy dependence, both of which are multiplied together to find the total yield. The angular yield has a polynomial dependence on incident angle making it easier to fit experimental data. The equations for the total yield, Y, in atoms/ion is given as

$$\begin{aligned} Y(e,\beta) &= S(E)g(\beta)C_6 \\ S(E) &= \sqrt{E}(1 - \sqrt{\frac{E_{th}}{E}})^{2.5} \\ g(\beta,E) &= C_1 + C_2\beta + C_3\beta^2 + C_4\beta^3 \end{aligned}$$

Where E_{th} is the threshold energy and is another input parameter defined by C_5. This model also uses a polynomial function in β instead of $\cos\beta$ such that the set of coefficients may not yield a sputter yield of zero at $\beta = 90°$.

`Pencil`

In the work by Pencil at NASA Glenn Research Center, an SPT-100 thruster was used as a contamination source [29]. There were a lot of data about coated CMX, a ceria-doped borosilicate which is used as a cover slide material for silicon solar cells (but no coefficients for CMX). The materials with coefficients from Ref. [29] are

- Normal incidence: Ag, Fe, Si, Borosil, (Quartz coverglass is studied as Si_2). The normal yield for Si must be multiplied by 0.57 to recover the normal yield for $Si0_2$.)
- Angular incidence: Ag, Fe, $Si0_2$ and Borosil. These are not implemented in the code.)

The general form for the `Pencil` model is as follows.

$$\begin{aligned} Y(E,\beta) &= S(E)g(\beta) \\ S(E) &= C_0 E^{0.25}\left(1 - \frac{C_1}{E}\right)^{3.5} \\ g(\beta) &= 1 + C_2(1 - \cos(C_3\beta))^{C_4} \end{aligned}$$

Sputter yields below the threshold energy, C_1, are zero.

3.5.8 Angular Distribution

`Cosine`

The `Cosine` angular distribution results in peak atom fluxes along the surface normal of the target. In the `Cosine` model, angles are randomly sampled using the equation [32]

$$\sin\phi = \sqrt{U_1}$$
$$\cos\phi = \sqrt{1-\sin^2\phi}$$
$$\theta = 2\pi U_2$$

Where U_1 and U_2 are random numbers. This results in the polar angular probability distribution function,

$$g(\phi) = \sin 2\phi$$

This model requires the sputtered particle speed V as an input.

`Zhang`

This model uses a modification to the redeposition algorithm developed by Zhang [30]. The particular feature of the modified Zhang model is that it assumes a data-driven fit for the $E*$ coefficient rather than using a physically motivated sputtering threshold energy. This is a relatively minor change so it is possible to use the modified Zhang sampling model to replicate the results of classical Yamamura sputtering by simply using the physically motivated $E*$ coefficient instead.

$$f(E,\beta,\alpha,\phi) = \frac{1}{1-\cos(\beta)\sqrt{\frac{E^*}{E}}}\frac{\cos(\alpha)}{\pi}\left[1-\frac{1}{4}\sqrt{\frac{E^*}{E}}\left(\cos(\beta)\gamma(\alpha)+\frac{3}{2}\pi\sin(\beta)\sin(\alpha)\cos(\phi)\right)\right]$$
$$\gamma(\alpha) = \frac{3\sin^2(\alpha)-1}{\sin^2(\alpha)} + \frac{\cos^2(\alpha)(3\sin^2(\alpha)+1)}{2\sin^3(\alpha)}\ln\left(\frac{1+\sin(\alpha)}{1-\sin(\alpha)}\right)$$

where β is the polar angle of incident particle with respect to surface normal, α is the polar angle of ejected particle with respect to the surface normal, and ϕ is the azimuthal angle of the ejected particle.

3.6 Electric Propulsion Plume Simulation 2: Cubit

Kari A. Kawashima

3.6.1 Introduction

This document is the second of three parts in creating and running an electric propulsion plume simulation using the Thermophysics Universal Research Framework (TURF). The steps to set up a plume simulation are:

1. Get or create a CAD drawing of a spacecraft.
2. Mesh the geometry and export in an abaqus file format.
3. Modify highlighted entries in `world.list`.
4. Update `component_TURF.txt`. Use component names from the surface mesh file, and specify material and potential for each component (if they are different from default values).
5. Identify relevant surface interaction and sputtering mechanisms, obtain coefficients for surface interaction models, and include in `surf_int.txt`.
6. Modify highlighted entries in `operations.sat.list`.
7. Run TURF simulation
8. Evaluate results.

Part two is a guide to creating and exporting a mesh geometry of a spacecraft to be used in a simulation (Steps 1 and 2). TURF requires a triangulated surface mesh in order to handle spacecraft geometries; while TURF can handle several different formats including abaqus, exodus, gambit, and off, only the abaqus format (.inp filetype) is fully supported in the current version. The abaqus format file can be exported from a CAD meshing program such as Cubit [21], which is used for the purposes of this example.

First a general overview of Cubit's functions and capabilities will be explained in the context of creating the initial CAD model. Depending on the kind of thruster and injection source, additional steps may be necessary to create an accurate geometry. Three generic spacecraft CAD files are provided in `/tutorial-TURF/Plume/CubitExample/` to give the user optional geometries to start with. Then, the surface mesh will be created, grouped and named appropriately, and the file will be ready to be exported for use in a simulation.

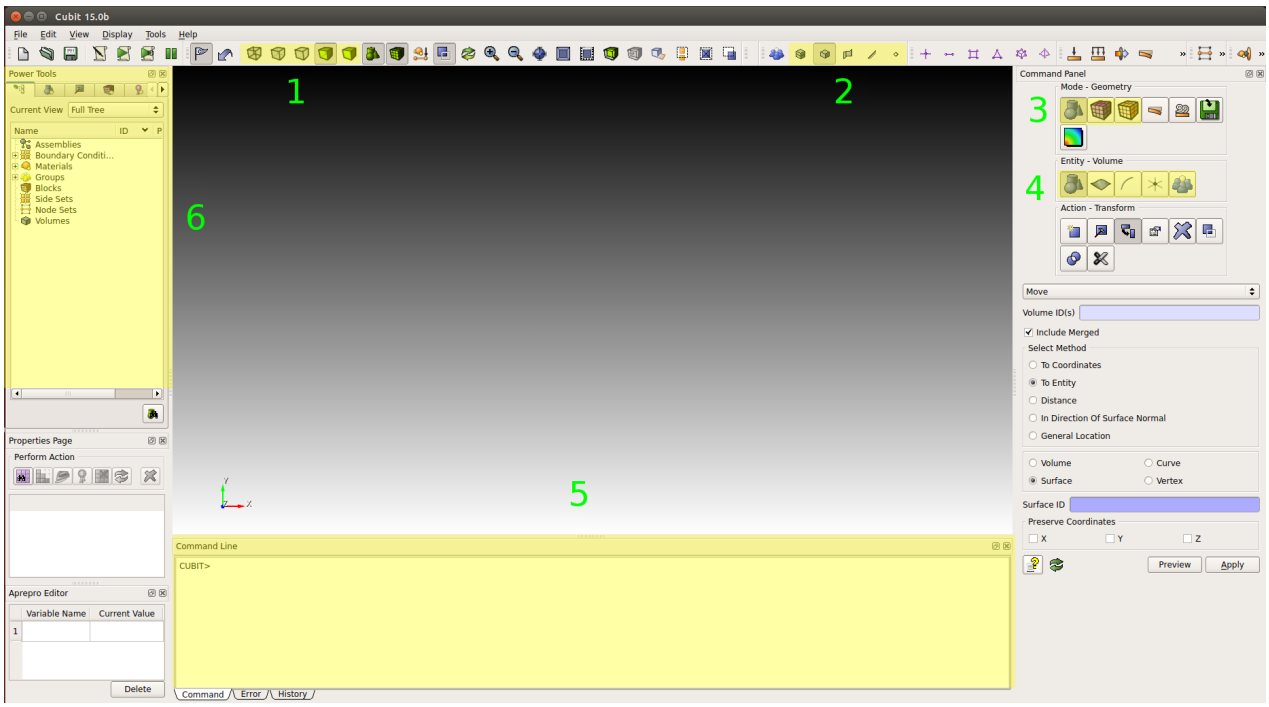


Fig. 3.12: The overall interface of Cubit

3.6.2 General Geometry

Cubit's interface layout is pictured in Fig. 3.12. The highlighted Area 1 is a set of visual settings buttons which change the transparency and presentation of volumes. Area 2 determines which kinds of entities can be selected when clicking on the geometry: volumes, surfaces, curves, or vertices. Area 3 highlights the three management modes used for this tutorial: geometry, mesh, and blocks. The buttons in Area 4 are similar in function to those in Area 2, but they are specific to the creation and manipulation of specific entities. The other useful tools are the command line located at the bottom of Cubit's interface shown in Area 5 and the tree summarizing the entities in the Power Tools toolbar on the left side as highlighted in Area 6.

To make a new volume, select the Volume management buttons in Areas 3 and 4, then click the first option "Create volumes." Options for the kind of volume to be created appear in a dropdown menu.

Manipulating an entity uses the Transform button under the Action toolbar. There are several particularly useful commands that will help make the desired geometry more quickly and accurately. One command is "copy," found as the last option under "create volumes" list, which allows the user to make a duplicate of selected volumes and move, rotate, etc., the copy without displacing the original volume. Another command is "reflect," found in the "transform" list, which moves an entity to its mirror position across a plane.

Also useful are the functions under the "webcutting operations" tab, which allow you to use entities to cut a volume into pieces. For example, Fig. 3.13 depicts one face of a cube "chopping" a hollow sphere into two half-dome volumes shown in Fig. 3.14. This is done by selecting the sphere as the "Webcut Target" and the cube as the "Cutting Tool," leaving separate halves of the sphere.

Note as well that commands can be entered in the command line directly. For the sake of accuracy, commands such as "`Measure between surface 3 5`" will respond with the measured distance between surface 3 and surface 5, and the graphics display will show a red 3D arrow to indicate which measurement was taken.

Once the desired satellite geometry has been created just as the one shown in Fig. 3.15, all the shapes must be combined to make a single volume entity and eliminate overlapping parts. The "Unite" function is under the "boolean operations for volumes" tab, as shown in Fig. 3.15. Type "all" into the Volume ID box. The result is a single volume, indicated by a change of the entire geometry to be the same color (unless there are unattached volumes floating in space). The united spacecraft geometry shown in Fig. 3.16 has been saved as `example1.cub` for the user's convenience, and can be found in `/tutorial-TURF/Cubit/CubitExample/`.

If the geometry must be altered after the volumes have been united, surfaces must be separated from the rest of the geometry so the changes can be made. "Separate" is found under the Modify toolbar for Surface selection. The chosen surfaces will form a separate "Sheet Body" volume which can be manipulated in the same manner as a normal volume.

3.6.3 Mesh

After the satellite geometry is properly united, its surfaces can be meshed. Select the "Specify meshing schemes and attributes" function under Mesh Mode, as shown in Fig. 3.17 (left) and change the meshing scheme from "Automatic" to "Trimesh" for best results. Tell Cubit to mesh all surfaces[20], click "Apply Scheme", then hit "Mesh." The results should resemble the geometry shown in Fig. 3.17 (right).

It is important to adjust the mesh resolution in order to maximize the TURF simulation efficiency. Too coarse surface mesh does not represent the curved surfaces accurately, while increasing the resolution will require more triangles to scan through in determining particle-surface intersections in TURF. The mesh resolution can be adjusted in Action "Define intervals and sizes."

3.6.4 Blocks

When all surfaces of the spacecraft geometry have been meshed, they must be organized into groups and named so they can be specified in TURF's input files. Cubit refers to such groups as Blocks. To create blocks, select the Exodus Mode in the command panel, select Block, and choose the Add function under the Manage operation. Give the new block an ID number not already in use and select the surfaces desired for the block. Group surfaces

[20] TIP: Instead of typing out all the surface IDs, *all* can be used in "Select Entities to Mesh" to point to all the surfaces.

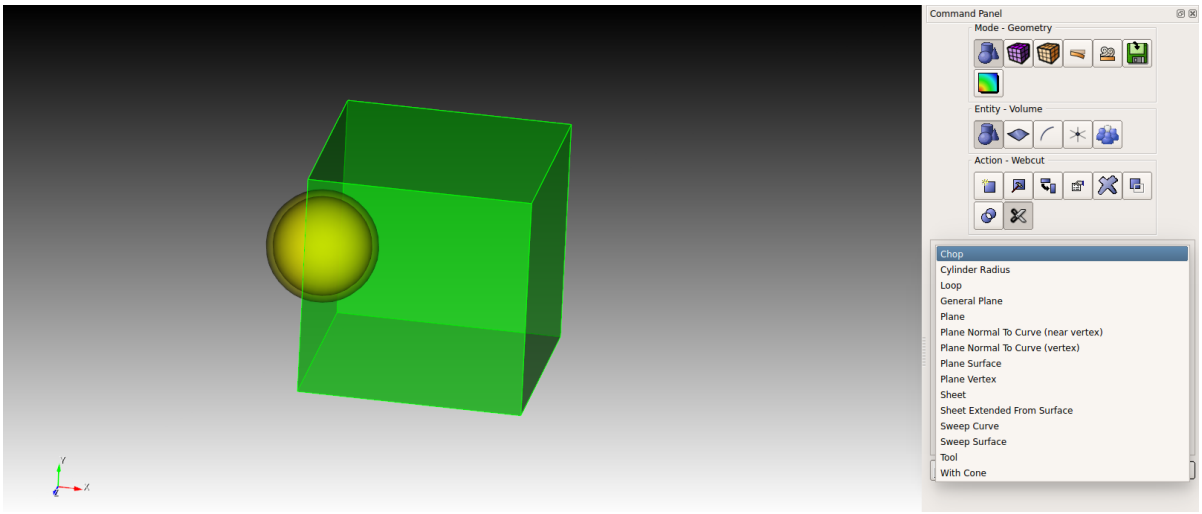


Fig. 3.13: Sphere and cube to be used in the ”chop” function

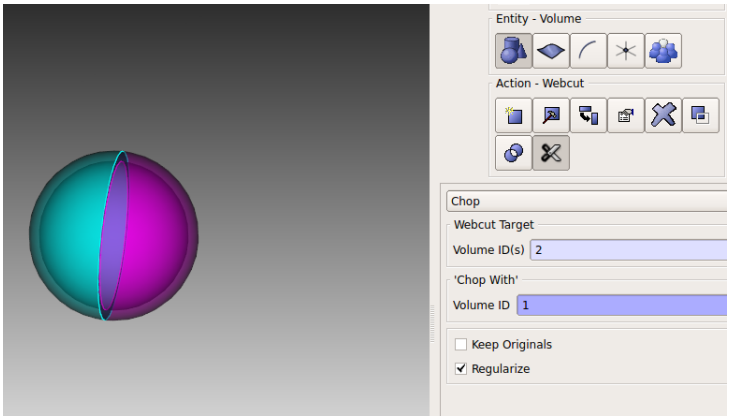


Fig. 3.14: The halves resulting from the chop operation shown in Fig. 3.13

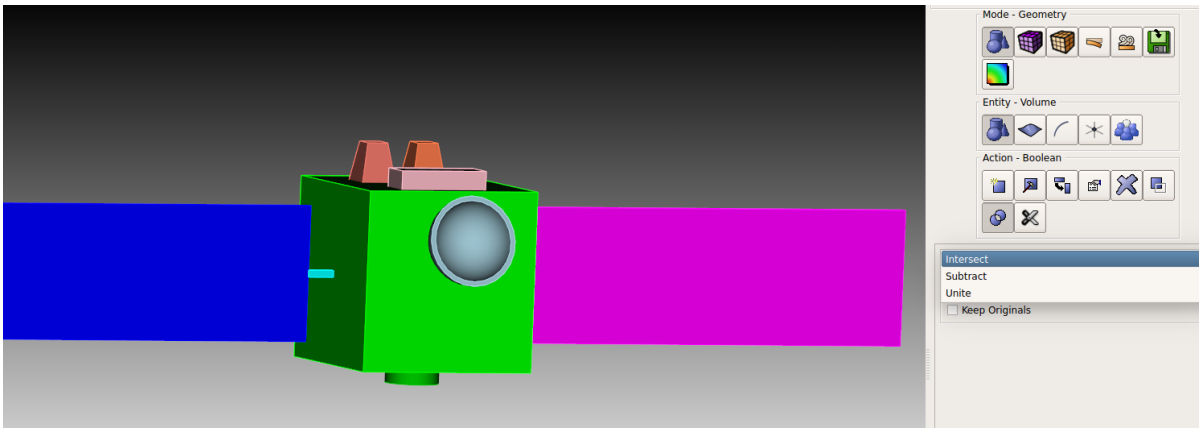


Fig. 3.15: Satellite geometry before volumes are united

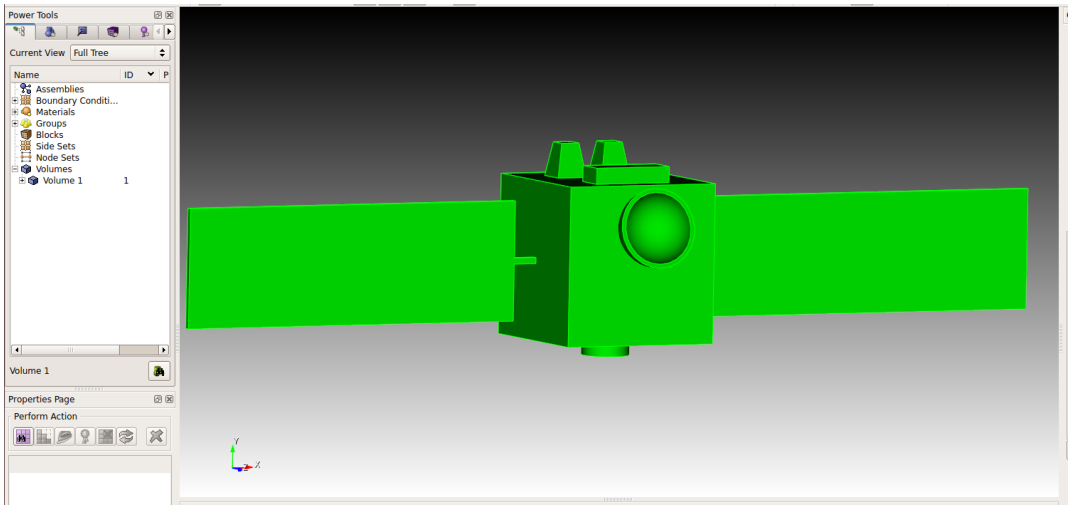


Fig. 3.16: United satellite geometry. Note the tree in the sidebar lists only one volume.

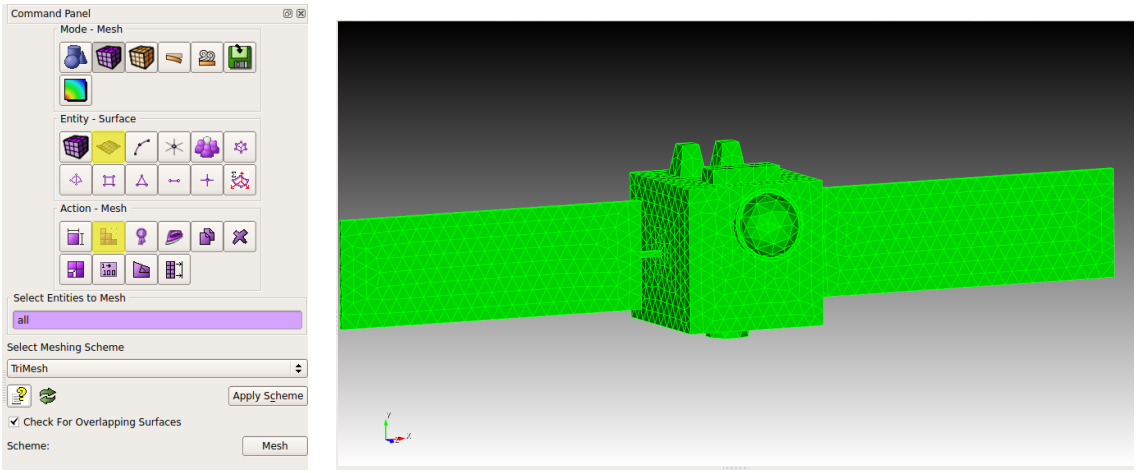


Fig. 3.17: Left: Apply the desired mesh to the surfaces of the satellite geometry. Right: Meshed satellite geometry.

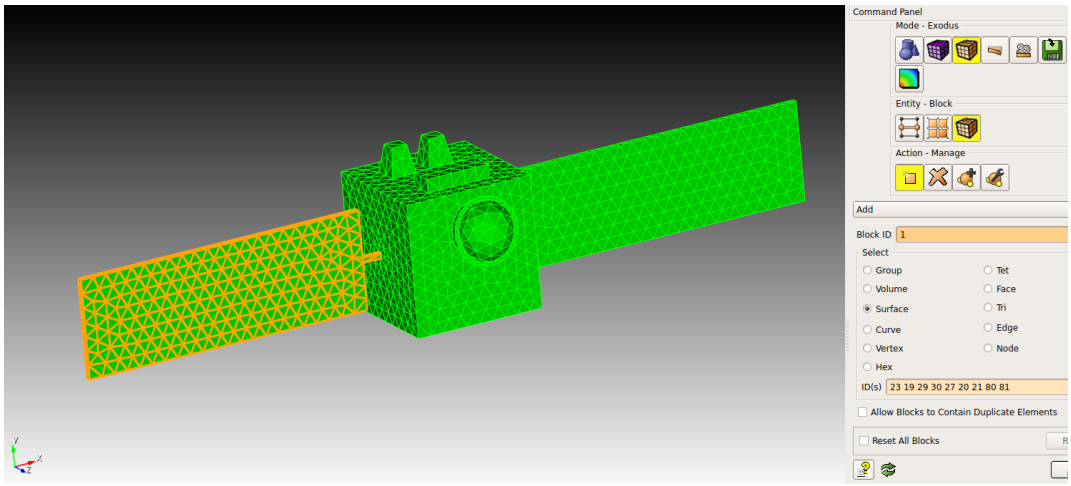


Fig. 3.18: The surfaces of one solar array are grouped into a block.

into blocks in a logical way (e.g. the surfaces of a solar array, the main body of the spacecraft, etc.), as shown in Fig. 3.18. When the block is created, change the name of the block in the component tree (Area 6 in Fig. 3.12) to describe the part (e.g. `SatelliteDish`). TURF's input files refer to these block names, so make them distinct and straightforward.

When creating blocks for the spacecraft, make separate blocks for the thruster's face and body. Doing so allows them to be assigned separate material properties for a simulation in the input files. It also allows injection sources and probes to be placed at specific locations, which is explained in greater detail in Part One of this tutorial.

The other groups can be as general or as specific as desired. It is beneficial to make separate blocks for satellite components which will either have distinct electric potentials for the simulation, or have a high level of concern with respect to damage due to ion sputtering and flyback.

3.6.5 Export

After the geometry is meshed, blocked, and named as necessary, the file is ready to be exported in an abaqus file format (.inp) compatible with TURF. A dialogue box with export options will pop up when the user hits save. Uncheck the option to export the file using Cubit IDs; the option will refer to the blocks by the ID numbers used in Cubit rather than the names the user has assigned, which is undesirable.

Make sure *all* surfaces of the geometry are in a block before the geometry is exported. If a surface is forgotten, there will be a hole in the exported mesh geometry that will cause the simulation to crash.

3.6.6 Advanced Examples

Ion Thruster Grid

For a spacecraft using an ion thruster, a special geometry consideration may need to be taken into account. Conventional ion thrusters of the size greater than 10 cm in diameter use curved grids in order to increase the structural integrity. Although the beam divergence is typically much less compared to Hall Thrusters, the ion beam still expands due to the potential field within the grids. On top of this, the curvature of the grids contributes significantly to the beam divergence.

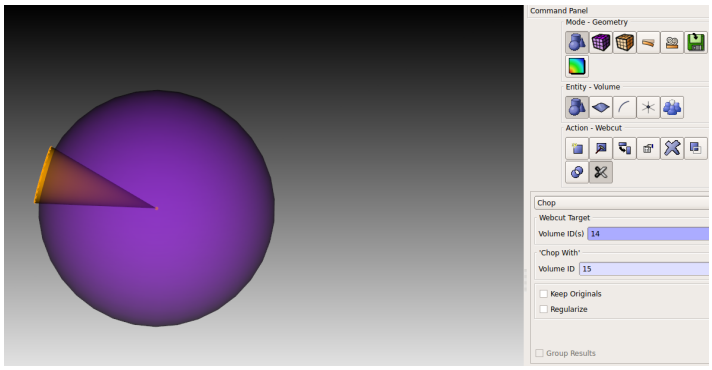
A surface with the appropriate curvature can be created by using the chop function with a sphere and a cone. When creating a cone, its bottom face needs to be extended past the sphere such that the cone volume can be chopped using the sphere volume. The process is displayed in Figs. 3.19(a) to 3.19(c).

Rescale, move, separate, and unite volumes and surfaces as necessary until the curve is properly assimilated into the satellite geometry. Then apply the surface mesh. An unmeshed geometry file with this thruster face has been saved as `example2.cub` for the user's convenience.

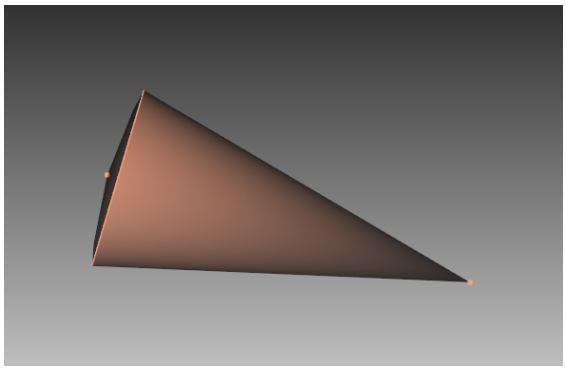
RPA Source

Particle injection using a Retarding Potential Analyzer (RPA) source occurs at a certain distance from the thruster face. Essentially, ions are injected from a sphere above the thruster face rather than the thruster face itself. Because of this injection sphere, there is a volume of empty space underneath it. The size of the injection sphere is determined by the size of the thruster and the maximum angle to which the RPA data is provided: its radius must be large enough to ensure particles are not injected within the thruster's geometry, but small enough so the injection sphere does not interfere with other components nearby.

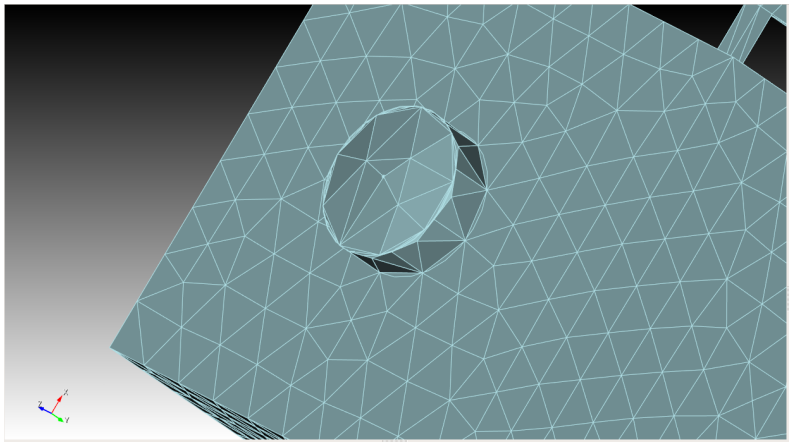
Due to the empty space underneath the spherical injection surface, the electric field establishes at the edge, possibly pulling slow ions toward the thruster face. In order to prevent this, a physical representation of the vacuum space under the injection sphere is useful for simulation purposes. This geometry is used solely in setting the potential within such that electric field does not establish within the region. Once an appropriate size for the injection sphere is determined, add a *slightly smaller* sphere centered at the thruster face to the geometry. However, the mesh of the sphere should be kept independent of the satellite's geometry. In other words, do not unite the spacecraft volume with the vacuum sphere volume. The correct setup is saved as `example3.cub`, and also shown in Fig. 3.20.



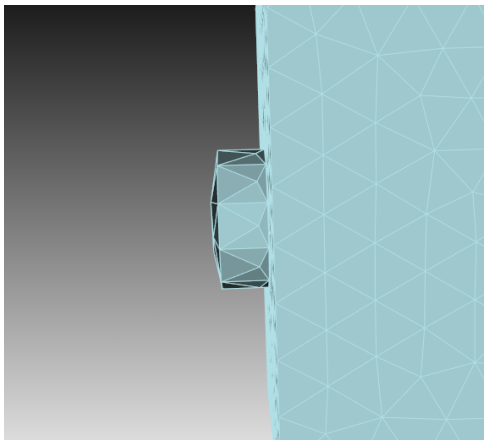
(a) Chop a slightly-oversized cone.



(b) The resulting volume. Note the slight curve on the cone's bottom face.



(c) The curved thruster face as part of the satellite's surface mesh.



(d) Another view of the thruster face to better display the slight curve of the surface mesh.

Fig. 3.19: Using Chop feature in Cubit.

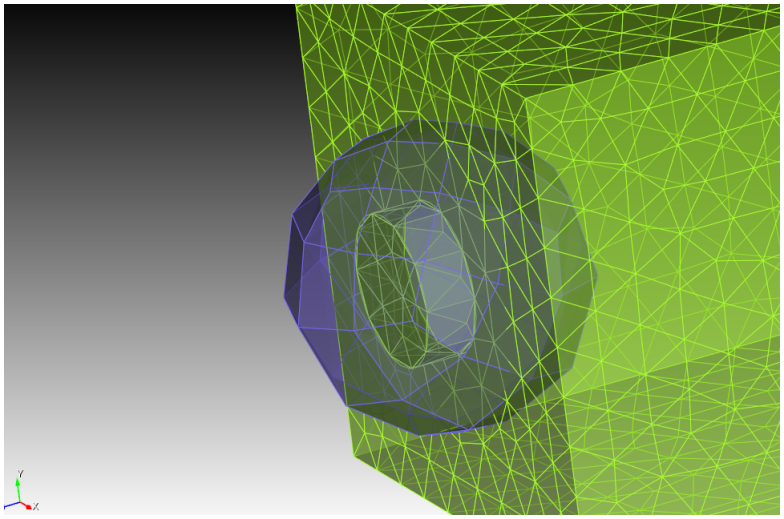


Fig. 3.20: A vacuum space sphere placed over a thruster face using an RPA source. Note that the sphere mesh overlaps the satellite mesh and is not a part of the entity.

3.7 Electric Propulsion Plume Simulation 3: ParaView

Kari A. Kawashima and Samuel J. Araki

3.7.1 Introduction

This document is the third of three parts in creating and running an electric propulsion plume simulation using the Thermophysics Universal Research Framework (TURF). The steps to set up a plume simulation are:

1. Get or create a CAD drawing of a spacecraft.
2. Mesh the geometry and export in an abaqus file format.
3. Modify highlighted entries in `world.list`.
4. Update `component_TURF.txt`. Use component names from the surface mesh file, and specify material and potential for each component (if they are different from default values).
5. Identify relevant surface interaction and sputtering mechanisms, obtain coefficients for surface interaction models, and include in `surf_int.txt`.
6. Modify highlighted entries in `operations.sat.list`.
7. Run TURF simulation
8. Evaluate results.

Part Three is a guide to displaying the simulation results in ParaView [19] (Step 8).

Table 3.19: Folders created by various TURF operations to output results.

File/Folder	TURF Operation	Type of Data
`UMesh.vtu`	`LogicalMeshSurfaceSugarcubeOp`	Surface mesh loaded to TURF
`UMesh_sugarcube.pvtr`	`LogicalMeshSurfaceSugarcubeOp`	Points to `UMesh_sugarcube_DOM.vtr` files output by each sub-domain
`UMesh_sugarcube_DOM.vtr`	`LogicalMeshSurfaceSugarcubeOp`	Sugarcube information
`Plot3D`	`LogicalFieldWriteVTKOp`	3D volume mesh field data
`PlotSurf`	`UFieldWriteVTKOp`	Surface mesh field data
`PlotTraj`	`MSPDistWriteVTKOp`	Particle data
`PlotXZ`	`LogicalFieldWriteVTK2DOp`	2D volume mesh field data
`Probe`	`ProbeStageWriteOp/ProbeFixedWriteOp`	Probe data

Once started the EP plume simulation from `tutorial-TURF/Plume/PlumeExample/`, various files and folders are created as listed in Table 3.19. The newly created folders (except `Probe`) contain files with extensions of `.pvts`, `.pvtu`, `.pvtp`, `.vtm`, `.vts`, `.vtu`, and `.vtp`. For the case of structured grid, TURF output one `.pvts` file and multiple `.vts` files, which the number of `.vts` files correspond to the number of sub-domains at a given iteration. The `.vts` file contains the data that belongs to a single sub-domain, and the `.pvts` file points to multiple `.vts` files, representing the whole simulation domain. There is an additional file with `.vtm` extension, which can be read in place of `.pvts` file.[21] The viewing of these files requires software compatible with the VTK format files; this tutorial uses ParaView [19] as the viewing software, but VisIt [20] is a program with similar functionality.

3.7.2 ParaView

When TURF runs a simulation, it will indicate when the imported surface mesh has been successfully sugarcubed.[22] This means that the geometry mesh has been fully imported and the output geometry file (`UMesh.vtu`) can be opened for viewing in ParaView, even before the time steps begin advancing. When opened and clicking "Apply" in the

[21] Loading `.pvts` file to the visualization software generally works, but some visualization software may work better with `.vtm` file.

[22] Once the surface mesh is loaded, the relationship between this mesh and the volume mesh is determined. This includes the number of surface elements contained within each volume mesh cell, cells that are inside and outside of geometries, etc. We call this process, "Sugarcubing" in this document.

sidebar, the satellite geometry should appear on the interface as shown in Fig. 3.21. The bounds of the domain can also be made visible by importing the `UMesh_sugarcube.pvtr` file. Figure 3.21 also highlights five areas in ParaView. Area 1 allows the user to display a specific frame of data in the simulation. Area 2 is the Active Variable Controls toolbar, Area 3 is the Common toolbar, Area 4 displays a tree listing all the existing entities, and Area 5 is the Properties sidebar.

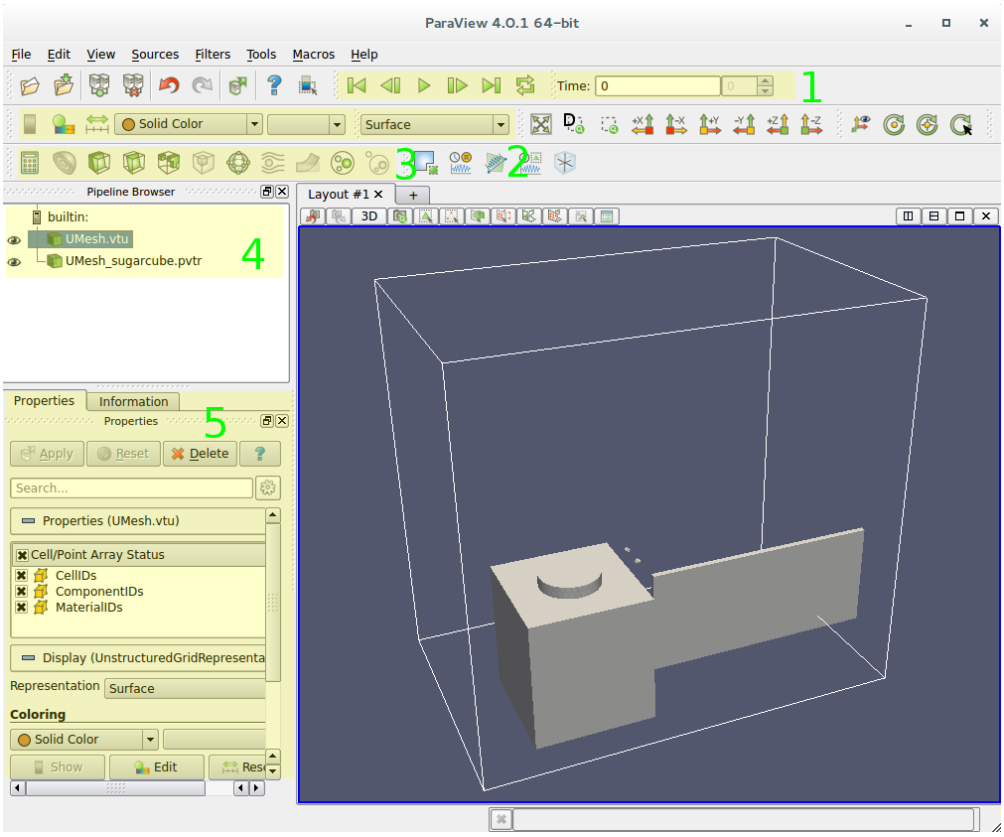


Fig. 3.21: Display of satellite geometry and a box representing the simulation domain. Some of ParaView's basic functions are highlighted in yellow.

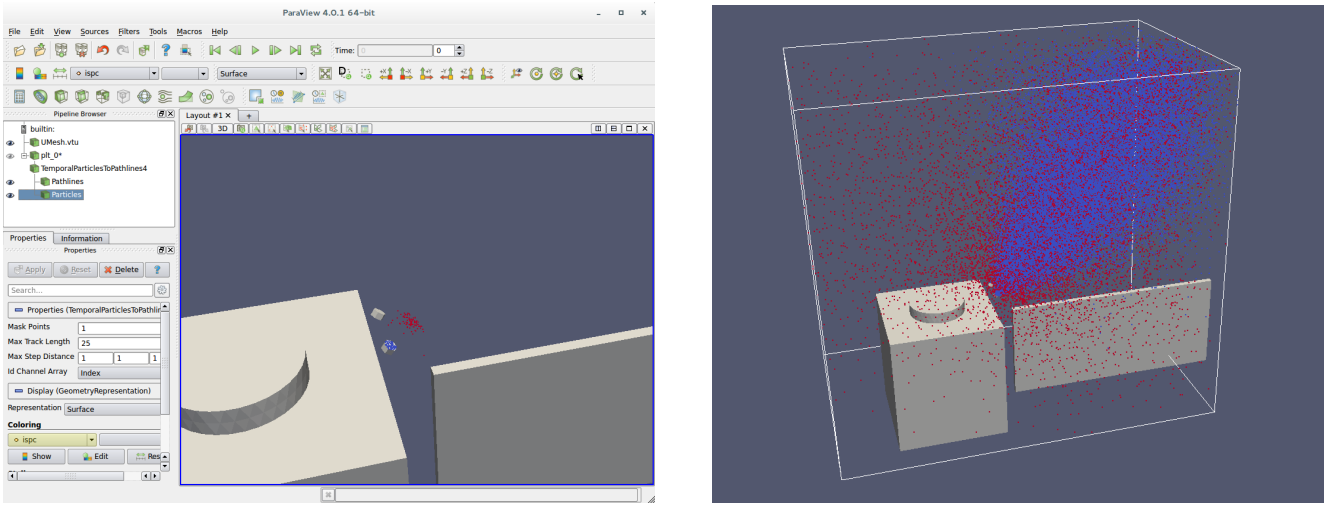
Particle Trajectory

To see the progression of particles for the duration of the simulation, open the `plt_..pvtp` collection of files inside the `PlotTraj` folder. With the `plt_0*` in Area 4 selected, go to the Filters tab in the taskbar and apply the "Temporal Particles To Pathlines" filter to the data. This filter should be able to be found under "Alphabetical" category under the Filters tab or by using "Search..." tool.

In the Properties bar, change the "Mask Points" value to 1 so the simulation does not inflate the number of particles in the display.[23] Also change "Id Channel Array" to "Index" to ensure the progression of particles is based on the correct position data. Small dots should appear on the face of the simulation's injection surface once the filter is applied (by clicking "Apply" button in Area 4). It is useful to change the colors of the particles depending on their gas species. Selecting Particles in the tree and changing the color display to "ispc" gives different colors to the neutral atoms and ions, as shown in Fig. 3.22.[24]

The buttons in Area 1 can be used to display the simulation frame-by-frame, skip to the end or beginning, or simply play them chronologically as a short video. For a clearer display of the particles, the visibility of the

[23] This value can be set to larger number if too many particles are displayed.
[24] As shown in Fig. 3.22(a), the color display is under "coloring" in Area 5 which is by default chosen to be "Solid Color".



(a) First frame.

(b) Last frame.

Fig. 3.22: Particle output from this tutorial simulation. Different colors are assigned to neutral (red) and charged particles (blue).

pathways can be turned off by clicking the eye icon next to its name in the tree, and vice versa. Figure 3.22(b) displays the particles at the end of simulation.

Surface Field Data

Data collected at the surface mesh, such as particle flux, can also be displayed. This can be particularly useful if there is concern that, for example, ion flyback could potentially damage mission-critical equipment mounted to the spacecraft.

Import the `pltSurf_..pvtu` file collection from the `PlotSurf` folder and hit "Apply". For clearer viewing of the flux, turn off the visibility of data from other files. Then, change the display to show the flux of the desired particle in Area 2 (e.g. `NPFLUX-IN_Xe+@g`). Go to the last time-step by clicking the forward button in Area 1. In case the color map does not cover the appropriate range for the flux data, click "Rescale" in Area 5. At this point, the surface flux of xenon ions should be displayed as shown in Fig. 3.23. Click "Show" in Area 5 or Area 2 to display the legend for the color gradient. "Edit" should also be used to manipulate the colormap as necessary

Volume Mesh Field Data

In order to visualize the volume mesh field data, first import the `plt_..vts` file collection from the `Plot3D` folder and hit "Apply". Select the corresponding `plt_0*` that appears in the tree, and then click the Slice button in the Common toolbar (Area 3). The command will create a plane within the domain at a specified location. For this geometry's orientation, a plane defined by the `Y Normal` shows the feature of plume. Change the Coloring from "Solid Color" to the field data of choice. Figure 3.24 shows the xenon ion density along the mid-plane of the simulation domain.

Alternatively, the files from the `plotXZ` folder can be imported instead of loading and slicing the 3D data. However, the plane of data output by `LogicalFieldWriteVTK2DOp` snaps to the plane of cell-centers. Therefore, for this particular example in which even number of cells is used along y-axis, the plane displayed in this folder would be half a cell off from the mid-plane.

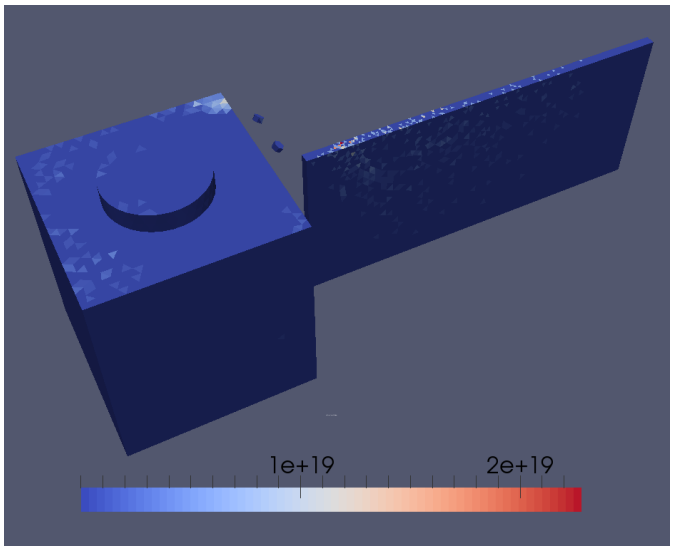


Fig. 3.23: The inward surface flux of Xe+ ions displayed on a spacecraft.

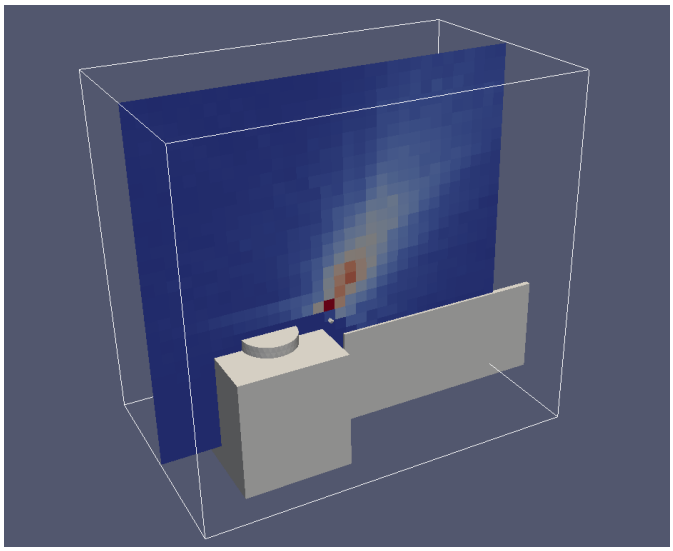


Fig. 3.24: Xenon ion density along the mid-plane of the simulation domain.

CHAPTER 4

TURF DEVELOPMENT

Robert Martin, Samuel J. Araki, and David Bilyeu

Contents

4.1 Extended Capabilities

Robert S. Martin and Samuel J. Araki

Beyond the material covered in the tutorials, several additional tutorials are currently being developed. These include a PIC and Fluid versions of the collisionless shock tutorial using the same problem setup (Section 2.6), ionizing breakdown using MCC collisions with particle merging and splitting, 1D Euler shock tube and 3D Euler shock-bubble fluid test cases, 3D DSMC bow-shock examples for both geometrically prescribed bodies as well as triangulated surface meshes, and a GPU accelerated version of the heatbath tutorial. Many of the Operations required for development of these tutorials are already complete but are not yet ready to be incorporated into the infrastructure release. The intent is to transition many of these Operations into the infrastructure in the future infrastructure releases. These operations may nevertheless be available to select developers as part of the Thermophysics Universal Research Framework development (TURF-DEV) package on a case-by-case basis. Brief descriptions of the operations are provided in Tables 4.1 and 4.2.

In addition to the TURF-DEV operations nearing incorporation in the infrastructure, another class of TURF-DEV operations are available that are still in experimental phases of development or are potentially too specific and research problem dependent to warrant inclusion in the infrastructure. Brief descriptions of these operations are included in Table 4.3 as reference for some future areas of development.

Table 4.1: Summary of operations included in TURF-DEV.

Module	Operation	Description
DSMC	`MSPDistDSMCCPUOp`	DSMC collision calculation
DSMC	`SPDistDSMCBoxICOp`	Uniformly distribute particles within a box
DSMC	`SPDistDSMCCPUOp`	Default RMS x&v random sign merge
Field	`BCFaceXtrapOp`	Extrapolate field over face direction
Field	`FieldArithmeticCoeffOp`	Perform arithmetic operation to two fields with two coefficients
Field	`FieldArithmeticConstOp`	Perform arithmetic operation to one field and one scalar value
Field	`FieldEvaluateEqnOp`	Evaluate a math equation and assign the result to field data
Field	`FieldMathFunctionOp`	Apply math function on one unstructure or structure field
Field	`FieldMathFunctionTwoArgOp`	Apply math function that requires two unstructure/structure fields
Field	`FieldScalarMulOp`	Multiply a field by a constant value
Field	`LogicalBCMirrorMoveInOp`	Move and add field ghost data over reflecting surface
Field	`LogicalBCPeriodicOneDomOp`	Copy ghost data for single domain periodic BC
Field	`LogicalDensityBoltzmannOp`	Find potential from Boltzmann relations
Field	`LogicalFieldBCEvaluateOp`	Adds one field variable to another
Field	`LogicalFieldBinaryCalcOp`	Binary field operations: C=A/B, C=A+B, or C=A-B
Field	`LogicalFieldBlendOp`	Blend two arrays over time
Field	`LogicalFieldEvaluateOp`	Adds one field variable to another
Field	`LogicalFieldFloorCeilingOp`	Apply a floor and ceiling value to a field
Field	`LogicalFieldLogOp`	Apply natural log to field
Field	`LogicalFieldMulOp`	Multiply or divide field by another field
Field	`LogicalGradientOp`	Calculate cell centered gradient
Field	`LogicalICConstantOp`	Set field values in box to constant
Field	`LogicalPoissonOp`	Red/Black Gauss-Seidel Poisson Solve
Field	`LogicalPoissonPeriodicStripOp`	Solve Poisson equation with periodic boundary conditions, requires cuFFT

Table 4.2: Summary of operations included in TURF-DEV (Continued).

Module	Operation	Description
Field	LogicalPoissonStrip1DOp	Red/Black 1D strip Poisson solve w/o transverse cells
Field	UFieldConstantICOp	Set unstructured field data to constant value
Field	UFieldICByComponentOp	Set UField data by component
Fluid	FluidConstantICOp	Set field parameters based off fluid definition
Fluid	FluidESPushOp	Advance fluid state with electrostatic forcing
Fluid	FluidSphereICOp	Set fluid parameters for cells within a sphere
Fluid	IntegrateFluxOp	Sum flux variables to advance fluid state
Fluid	LinearSemiLagrangianOp	Linear advection using semi-Lagrangiaan advance
Fluid	RoeFluxCalc2Op	Calculate 2nd order Roe flux
Fluid	RoeFluxCalcHLLE2Op	Calculate 2nd order Roe flux with HLLE2 limiter
Fluid	RoeFluxCalcHLLEPOp	Calculate 2nd order Roe flux with HLLEP limiter
Fluid	RoeFluxCalcOp	Calculate 1st order Roe flux
Fluid	TVDFluxCalcOp	Calculate total variation diminishing flux
Geometry	MSPDistSugarcubeSurfBCMoveOp	Perform particle surface reflection
Geometry	SPDistBCSurfBoxICOp	Add particles outside sugarcubed surface only
Geometry	SPDistSugarcubeBCMoveOp	Linear particle advance with specular reflection of marked cells
Particle	MSPDistEmptyOp	Set particle count to be zero
Particle	MSPDistSortTwoOp	Sort MSPDist for cellID then speciesID
Particle	MSPDistVSortOp	Sort particles by cell and velocity octant
Particle	SPDistBCBoxICOp	Initialize particles uniformly in physical box except flagged cells
Particle	SPDistBCDiffuseOp	Apply diffuse reflection
Particle	SPDistBoxICOp	Initialize particles uniformly in physical box
Particle	SPDistDirectCellMergeOp	Default RMS x&v random sign merge
Particle	SPDistDirectCellSplitOp	Default RMS x&v random sign split
Particle	SPDistESPhiCNPushOp	Crank Nicolson electrostatic particle potential push
Particle	SPDistESPhiNCNPushOp	Nonlinear Crank Nocolson electrostatic particle potential push
Particle	SPDistESPhiPushOp	Explicit electrostatic potential particle push
Particle	SPDistFractionalSplitBoxOp	Split particles in box in place by fraction on w
Particle	SPDistMCCOp	Monte Carlo Collision Operator
Particle	SPDistPerturbedCellIDOp	Particle cell ID with added perturbation for smoothing
Particle	SPDistScaleWeightByFieldOp	Scales SPDist density FIELD_BASE by FIELD_DELTA
Particle	SPDistSortedStatToFieldOp	Accumulate cell velocity moments from sorted distribution
Particle	SPDistTemperatureToFieldOp	Calculate cell temperature from distribution
Particle	SPDistToEMFieldOp	Accumulate cell charge and current from distribution
Particle	SPDistToFieldLinear1DOp	Accumulate 1D linear weight charge to cells
Particle	SPDistToFieldLinearOp	Accumulate linearly weighted charge to cells
Particle	SPDistVSortOp	Sort particles by cell and velocity octant
Plotting	LogicalFieldReadVTKOp	Read LogicalField data from LogicalFieldWriteVTKOp output
Plotting	LogicalFieldStatWriteVTKOp	Write accumulated statistic data to VTK file
Plotting	LogicalFieldWriteVTKUOp	Write unstructured field data to .vtu file
Plotting	LogicalVlasovFluidWrite1DProbeVDFOp	Write particle VDF from particles in probe region
Plotting	ProbePointOp	Outputs field data time series from a given point
Plotting	UPCWriteVTKOp	Outputs UPC PIC data to vtk file
SourceModel	MSPDistSourceCosineOp	Cosine distribtuion particle source for rarefied flows
SourceModel	SPDistNormalMaxwellianStreamOp	Maxwellian stream source from triangulated surfaces
Surface	MSPDistOutgassingOp	Outgassing from surfaces
Utility	ConfigurationTestOp	Collects accuracy test results for operator combinations
Vlasov	LogicalBCPeriodicOneDomVlasovOp	Periodic BC for Vlasov
Vlasov	LogicalFluidToLogicalVlasovFluidOp	Create Vlasov VDF from Fluid Variables
Vlasov	LogicalVlasovFieldVolumetricMulOp	Multiply or divide Vlasov data by cell volumes
Vlasov	LogicalVlasovFluidSetOp	Set Vlasov field data to constant
Vlasov	SPDistToLogicalVlasovFluidOp	Accumulate particle weights to Vlasov VDF

Table 4.3: Summary of additional experimental (E) and research (R) operations included in TURF-DEV.

Module	Operation	Description
DSMC	`SPDistDirectDSMCCellMergeOp`	(R) Default RMS x&v random sign merge
DSMC	`SPDistDSMCCPUB2BOp`	(R) Default RMS x&v random sign merge
Field	`LogicalICFunctionOp`	(E) Set field value using external function
Field	`CalculateHallMHDOp`	(R) Calculate perpendicular and cross-flow components of mobility.
Field	`CalculateHallMHDRHSOp`	(R) RHS for Ohm's law solve (IEPC-2015-314)
Field	`CalculateIonizationRateOp`	(R) Calculates ionization rates from field data.
Field	`CalculateMobilityComponentsOp`	(R) Evaluate Mobility Components (IEPC-2015-314)
Field	`ConstantAnnulusPotentialOp`	(R) Apply annularly symmetric potential to field
Field	`ConstantSphericalPotentialOp`	(R) Apply spherically symmetric potential to field
Field	`ConstantWellPotentialOp`	(R) Apply constant parabolic well potential to field
Field	`SetEFieldOp`	(R) Impose a constant vector everywhere
Geometry	`LogicalMeshSurfaceBBoxSplotchOp`	(E) Mark surface triangle bounding box on cells
Particle	`SPDistBCChildLangmuirOp`	(E) Child Langmuir surface emission
Particle	`SPDistBCSCLOp`	(E) Space charged limited flux boundary
Particle	`SPDistBCSecondaryOp`	(E) Secondary emission boundary condition
Particle	`SPDistBCTransOp`	(E) Linear Convection BC for Neutrals (IEPC-2015-314)
Particle	`SPDistCFEBCOp`	(E) Fowler-Nordheim cold field emission BC
Particle	`SPDistCopyPosBoxICOp`	(E) Initialize particles with VDF but positions from second dist
Particle	`SPDistDirectCellMergeXxVOp`	(E) Angular momentum preserving merge
Particle	`SPDistESPhiPushVerletHalfOp`	(E) Verlet electrostatic potential particle push
Particle	`MSPDistMCCElasticTableOp`	(R) Apply MCC elastic collision
Particle	`SPDistDirectCellMergeMixOp`	(R) Position sign from xv moment merge
Particle	`SPDistDirectCellMergeMixXVOp`	(R) Position sign from xv moment merge
Particle	`SPDistDirectCellMergePCAOp`	(R) Principle Component Eigendecomposition Merge
Particle	`SPDistDirectCellSplitMixOp`	(R) Position sign from xv moment split
Particle	`SPDistDirectCellSplitMixXVOp`	(R) Position sign from xv moment split
Particle	`SPDistDirectCellSplitPCAOp`	(R) Principal component analysis split
Particle	`SPDistDirectCellSplitXxVOp`	(R) Angular momentum preserving split
Particle	`SPDistESPhiNCNSpherePushOp`	(R) Nonlinear Crank Nicolson in spherical ES-potential push
Particle	`SPDistOrbiterICOp`	(R) Period synchronized particle initial condition
Plotting	`LogicalFieldCatalystOp`	(E) Kitware Catalyst plotting connection
Utility	`ConstantOperatorOp`	(E) Work to allow generic constant parsing
Utility	`FromGPUOp`	(E) Recursive GSObject transfer from GPU
Utility	`ToGPUOp`	(E) Recursive GSObject transfer to GPU
UPC	`SPDistConstantVlasBCOp`	(R) Constant boundary conditions for VlasovPIC
UPC	`SPDistConstantVlasICOp`	(R) Constant initial conditions for VlasovPIC
UPC	`SPDistVlas2wOp`	(R) Density to weights for VlasovPIC
UPC	`SPDistVlasDensityToFieldOp`	(R) Integrate density to field data for VlasovPIC
UPC	`SPDistw2VlasOp`	(R) Weight to density for VlasovPIC
Vlasov	`LogicalVlasovFluidFunctionICOp`	(E) Initialize Vlasov data using external function
Vlasov	`LogicalVlasovBGKOp`	(R) Apply the BGK collision operator to a Vlasov variable

4.2 Weak Landau Vlasov Test Case

David Bilyeu

4.2.1 Introduction

This example sets up the Landau dampening test case in the Thermophysics Universal Research Framework (TURF). This tutorial assumes that the user has a basic understanding of how to setup a TURF simulation, for a review please refer to the heatbath tutorial. The Landau test case is a standard simulation used to determine the accuracy of a Vlasov simulation. By default the simulation is setup to perform the "weak" or "linear" Landau dampening simulation but can be switched to the "strong" or "non-linear" case by changing a parameter in the operations.fluid.listfile.

The necessary files can be found in `tutorial-TURF/TURF-DEV/WeakLandau/Vlasov` and consist of

1. world.list
2. operations.fluid.list
3. Makefile
4. landau_ic.cu

The world.list and operation.fluid.list file are similar to other examples but the Makefile and landau_ic.cu file are new for this simulation. The landau_ic.cu file is a c/c++ sourcecode file that contains an externally defined c function, used to set the initial conditions for the simulation. The Makefile contains the proper commands to build a shared library that will be linked with TURF during runtime. As a result users are required to run the make command before running this simulation.

Also new to this tutorial is the Vlasov variable. These variables exist in a six-dimensional phase-space, having values that are dependent on both its physical location, x,y,z, and its velocity, Vx, Vy, Vz. Unlike a field variable that are defined in the world.list file, these variables are defined in the operations.fluid.listfile.

4.2.2 Problem Setup

In addition to the standard requirements in the world.list file the origin and delta values for the velocity mesh also needs to be defined. This is accomplished via:

```
VELOCITY_ORIGIN = (0.0,-0.5,-0.5)
VELOCITY_DELTA = (0.09375,1.0,1.0)
```

It is important to note that the delta values for the "y" and "z" component are unity. This is a requirement for a 1D1V Vlasov simulation to work properly.

This simulation uses 12 different filed variables,

```
FIELDS = [rhoE, rhoI, Ex, Ey, Ez, phi, rho_source, rho_total]
FIELDS = [Vmean_totalx, Vmean_totaly, Vmean_totalz, Temperature_total]
```

These variables are, electron density, ion density, the x, y, z components of the electric field, electric potential, temporary variable, and total density respectively. The second line contains the mean velocity in the x, y, and z direction and the temperature.

The following operators are broken up into three stages, INITIALIZE, PREOP, and MOVE and are listed in Table 4.4.

4.2.3 Operations

This section explains how to solve this problem using Vlasov methods within TURF. It is assumed that the reader has a basic understanding of how to setup a simulation in TURF and only the information relevant to Vlasov and this simulation in particular are detailed. The operations.fluid.listfile are broken up into three stages as defined in `world.list`.

Table 4.4: Summary of operations listed for the Landau Dampening Vlasov example.

Stage	Operation	Description
INITIALIZE	`CreateVlasovVariableOp`	Create new Vlasov fluid variable
	`LogicalVlasovFluidFunctionICOp`	EXPERIMENTAL - Initialize Vlasov data using external function
	`LogicalFieldSetOp`	Set field values to constant
PREOP	`LogicalVlasov2DWriterOp`	Exports a 2D phase-space plot
	`LogicalFieldWriteVTKROp`	Exports the field data in .vtr format
	`VlasovMetricsOp`	Exports Vlasov metric data for mass and energy conservation
MOVE	`LogicalBCPeriodicOneDomVlasovOp`	Periodic BC for Vlasov
	`Vlasov1D1VSLOp`	Advects a Vlasov variable using the Semi-Lagrangian method
	`LogicalVlasovCalcFluidVariablesOp`	Calculate field variables given a velocity distribution
	`LogicalFieldSetOp`	Set field values to constant
	`LogicalFieldSetOp`	Set field values to constant
	`LogicalFieldAddOp`	Adds one field variable to another
	`FieldScalarMulOp`	This operator multiples a field by a constant value
	`LogicalFieldVolumetricMulOp`	Multiplies or divides by cell volume
	`LogicalPoissonStrip1DOp`	Red/Black 1D strip Poisson solve w/o transverse cells
	`LogicalGradientCellCenterOp`	Calculates the gradients of a field vector
	`LogicalBCVlasovExtrapolateOp`	Sets a velocity boundary conditions to extrapolation
	`Vlasov1D1VSLOp`	Advects a Vlasov variable using the Semi-Lagrangian method
	`LogicalBCPeriodicOneDomVlasovOp`	Periodic BC for Vlasov
	`Vlasov1D1VSLOp`	Advects a Vlasov variable using the Semi-Lagrangian method

Stage: Initialization

In the initialize stage it is necessary to set the initial conditions which includes defining a new phase-space variable. The phase space variable, `vFe`, is defined using:

```
DEFINE OPERATION
   TYPE = CreateVlasovVariableOp
   DATA_NAME = VlasovFluidData
   VBOX_LO = (-6.0,-0.5,-0.5)
   VBOX_HI = ( 6.0, 0.5, 0.5)
   SPECIES_NAMES = vFe
   SPECIES_COMPOSITION = NONE
   M = 1.0
   Z = -1
   VBOX_NGHOST = [3 0 0]
END OPERATION
```

The keys are `DATA_NAME` which stores a group of Vlasov variables and other relevant data, `VBOX_LO` and `VBOX_HI` which sets the velocity bounds, `SPECIES_NAMES` which gives names to the Vlasov variables, and `VBOX_NGHOST` sets the number of ghost cells in each velocity direction. The last set of variables associates a species with it's molecular weight and charge. This is accomplished in one of two ways, the first is to provide the elemental makeup

along with its charge, e.g. Ar+@g, CH3@g, ..., through the SPECIES_COMPOSITION key, the second is to set the SPECIES_COMPOSITION to NONE and provide it's atomic mass weight, M, and charge Z. If multiple species are defined than SPECIES_COMPOSITION for each species is required. For example

```
DEFINE OPERATION
    TYPE = CreateVlasovVariableOp
    DATA_NAME = VlasovFluidData
    VBOX_LO = (-6.0,-0.5,-0.5)
    VBOX_HI = ( 6.0, 0.5, 0.5)
    SPECIES_NAMES = vFe, vFAr, vFAr+, vFunky
    SPECIES_COMPOSITION = e-@g, Ar@g, Ar+, NONE
    M = 18.0E-19
    Z = 0
    VBOX_NGHOST = [3 0 0]
END OPERATION
```

This operator has optional parameters including, VELOCITY_DELTA and VELOCITY_ORIGIN, these parameters defines the velocity spacing and the velocity origin. If they are not provided then the default values defined in the world.list file are used instead. LOCAL_VELOCITY_COORDINATE can also be set to True which allows for a local definition of the velocity coordinates.

The next operator used in this simulation is the Logical Vlasov Fluid Function IC Op. This is an externally defined c function that provides more flexibility in providing initial conditions. A Makefile is provided to that will compile the function into a shared library which will be loaded during run time.

```
DEFINE OPERATION
    TYPE = LogicalVlasovFluidFunctionICOp
    DATA_NAME = VlasovFluidData
    EXT_LIB = libinitial.so
    EXT_FCN = weaklandau
    VARIABLE = vFe
    FCN_ARGS = (al= 0.01, k= 0.5) #Weak
    NUMBER_OF_DIMENSIONS = 1
END OPERATION
```

The keywords in this operations are: DATA_NAME which stores the group of Vlasov variables, EXT_LIB which is the name of the shared library, EXT_FCN which is the name of the function in the shared library, Variable is the Vlasov variable name, FCN_ARGS is a list of arguments to be provided to the function, and NUMBER_OF_DIMENSIONS which is the number of velocity dimensions. In the right hand side of the FCN_ARGS keyword are first parsed by commas, then parsed by equal signs. Therefore the al and k will be removed leaving a list containing 0.01 and 0.5.

The function for this operator takes the form,

```
extern "C" {
double weaklandau(double x, double y, double z, double vx, double vy, double vz,
                  int Nin, double *Args){
    double rtn;
    double PI = 2.0*asin(1.0);
    double al = Args[0];
    double k = Args[1];
    double tmp = 1.0 + al*cos(k*x);

    rtn = tmp/sqrt(2.0*PI)*exp(-vx*vx/2.0);
    return rtn;
}
}
```

The parameters in the function call are: the x, y, and z coordinates, x, y, and z velocity coordinates, length of double array, and the double array. Note that the length of the double array will be determined by the operator and is not provided by the user. It is recommended but not required to use the same compiler that was used to compile TURF.

The next three operators are in the `PREOP` stage. These operators include `LogicalVlasov2DWriteOp`, `LogicalFieldWriteVTKROp`, and `VlasovMetricsOp`. Since `LogicalFieldWriteVTKROp` has been defined elsewhere it will be skipped in this discussion.

The LogicalVlasov2DWriteOp operator will take two directions in phase-space and plot them in a two-dimensional field.

```
DEFINE OPERATION
    INCLUDE_GHOST = false
    TYPE = LogicalVlasov2DWriterOp
    DATA_NAME = VlasovFluidData
    FILE_HEAD = weak_landau/phase_
    FIELD_NAME = vFe
    SKIP = 10
    SPACE_CORD = (0.0, 0.0, 0.0)
    VELOCITY_CORD = (-5.0, 0.0, 0.0)
    X_PLOT_DIRECTION = X
    Y_PLOT_DIRECTION = VX
    BINARY = true
    RUN_AT_INIT = true
END OPERATION
```

The keywords in this simulation are: `INCLUDE_GHOST` which plots the ghost cells, `DATA_NAME` the object that holds the Vlasov variables, `FILE_HEAD` the name of the out put files, excluding the time step and file extension, `FIELD_NAME` the name of the Vlasov variable, `SKIP` how many iteration to skip in between saving, `SPACE_CORD` a spatial coordinate that exist in the plane to save, `X_PLOT_DIRECTION` the phase space coordinate to plot along the horizontal, x, axis, `Y_PLOT_DIRECTION` the phase space coordinate to plot along the vertical, y, axis, `BINARY` save the VTK file in binary format, and `RUN_AT_INIT` which will run this operator during the init stage. An optional unused parameter is `PRINT_BEFORE_UPDATE` which adjust the iteration number based on if the plotting is done before or after the variable is updated. It is `False` by default.

The `VlasovMetricsOp` operators provides metrics over the entire domain for a single Vlasov species. It includes, iteration, time, density, entropy, energy, electric field, and momentum in the x, y, and z direction. The density, entropy, and momentum are calculated by taking moments of the velocity distribution, than integrating over physical space. The energy is the sum of the kinetic energy, via moments of the velocity distribution, and the electric field energy. The electric field is calculated by integrating the square of the electric filed over space. To improve accuracy the moment calculation uses Kahan summation to reduce round off errors, this was found to have a noticeable impact quality of the simulation as the velocity domain increased in size. The operator is setup via:

```
DEFINE OPERATION
    TYPE = VlasovMetricsOp
    PHASESPACE_TYPE = VlasovFluidData
    SPACE_TYPE = FieldData
    PHASESPACE_NAME = vFe
    DENSITY_NAME = rhoE
    E_FIELD_PREFIX = E
    E_FIELD_DIRECTIONS = [x, y, z] ## Need all three for this to work
    FILE_NAME = weak_landau/norms.csv
    SKIP = 1
    RUN_AT_INIT = false
END OPERATION
```

PHASESPACE_TYPE the object that holds the Vlasov variable, SPACE_TYPE the object that holds the field variables, PHASESPACE_NAME the Vlasov variable, DENSITY_NAME [UNUSED] the field name that holds the density variable, E_FIELD_PREFIX the prefix for the electric field, E_FIELD_DIRECTIONS the directional suffixes for the x, y, and z directions of the electric field, FILE_NAME the name of the out put files, excluding the time step and file extension, SKIP the number of iteration to skip in between saves, RUN_AT_INIT run the apply method during the init stage.

The next stage is MOVE. This is the final stage of the simulation and sets up the evolution of the simulation in time. Due to the complexity of the stage a more detailed overview is provided before a more in depth explanation of each of the operators. The stage consists of the following operators: LogicalBCPeriodicOneDomVlasovOp, Vlasov1D1VSLOp, LogicalVlasovCalcFluidVariablesOp, LogicalFieldSetOp, FieldScalarMulOp, LogicalFieldVolumetricMulOp, LogicalPoissonStrip1DOp, LogicalGradientCellCenterOp, and LogicalBCVlasovExtrapolateOp. Most of these operators are unique to Vlasov simulation and will be discussed in detail. Only the LogicalFieldSetOp, FieldScalarMulOp, and LogicalFieldVolumetricMulOp will be skipped.

The overall strategy for this simulation is to use a second order directional splitting algorithm to decouple the advection in the x and Vx directions. Each direction is solved using a fifth order semi-Lagrangian with WENO weighting to advance the solution in each of the directions. The electric field is assumed to be static so the Poisson Equation is used. In this case the ions are assumed to be constant and are not modeled. To start, the solution advances for half a time step in the x direction by applying periodic boundary conditions Vlasov1D1VSLOp and the semi-Lagrangian method Vlasov1D1VSLOp. The next step is to calculate the electric filed using the updated Vlasov variable. The first step in the process is the calculation of the field variable, LogicalVlasovCalcFluidVariablesOp, which takes moments of the distribution function to calculate density, velocity, and temperature at each spatial location. The second step is to initialize rho_source and phi, which is a temporary variable and electric potential respectively, using LogicalFieldSetOp. This is followed by calculating the electron charge density, rhoE, by adding the ion density, rhoI, and total density, rho_total , via the LogicalFieldAddOp op. The next steps multiples the electron charge density by the permittivity of free space and by the cell volume to get the total charge instead instead of the charge density. It is necessary to multiple by the permittivity to counter act dividing by this value in the LogicalPoissonBoltzmannStrip1D op. This is only needed for certain test cases as they normalize the variables. The next step is to solve the Poisson equation to calculate the electric potential, phi, via the LogicalPoissonBoltzmannStrip1D. And is followed by taking the negative gradient of phi to find the electric filed, Ex, Ey, and Ez. The next step in the calculation is to advance the solution by a full time step in the Vx direction. This is accomplished by first applying the periodic boundary condition Vlasov1D1VSLOp in the Vx direction followed by applying the semi-Lagrangian method in the Vx direction Vlasov1D1VSLOp. Both of these are done over a full time step. To finish a single time step requires advancing the solution for half a time step in the x direction. This is accomplished by applying periodic boundary conditions Vlasov1D1VSLOp and the semi-Lagrangian method Vlasov1D1VSLOp.

APPENDIX A

PROGRAMMING STYLE GUIDE

David Bilyeu

Contents

A.1 Introduction

The purpose of this document is to provide a set of guidelines to ensure uniformity across the entire code. This will ease the difficult task of code maintenance especially when the original developer has found greener pastures. These guidelines are not designed to restrict functionality of the code and in cases where the established guidelines interfere with the functionality of the code the guidelines should be ignored. This document will be split into three main sections; the first deals with the appearance of the code, the second addresses code compliance, and the third deals with software testing.

A.2 Coding Standards

A.2.1 Naming Convections

When editing an existing file a programmer should follow the naming convections of the existing file when starting an new file the following convections should be used.

- **Classes:** Its name will be CamelCase, should be descriptive, no abbreviations and should be short, i.e. 2-3 words.
  - Classes that define an operator must end with Op, e.g. SpecularPICBCOp.

  - Classes that operate on logical fields should begin with Logical. Logical fields are those that operates on three-dimensional structured data, i.e., SMesh meshes. E.g., LogicalFieldNormOp would calculate the norm of a field variable that exists on an SMesh, preferably written to handle curvilinear meshes.
  - Classes that operate on both a UMesh and SMesh field data cannot contain the word Logical, e.g. FieldNormOp would compute the norm of either a variable stored on either a UField or SField.

- **Functions:** Its name will be lower camel case, descriptive and short, e.g. thisIsAFunction
- **Macros/enums:** and other constants name is ALL_CASSES_WITH_UNDERSCORES. The use of macros to define constants, e.g. PI for 3.141592653589, is discouraged because they have a global scope and cannot be encapsulated.
- **Variables:** Their name will be all lowercase_with_underscores (when necessary) and will be defined in the scope in which it is used.
  - Member and function variables should be greater than 3 letters and have a name that is descriptive. There is nothing wrong with long, descriptive variable names, e.g., ion_temperature is favorable over ion_temp or i_temp.
  - Iterators should start with i-n and be less than 4 letters
  - Temporary should start with "tmp_" or "itmp" and should be set and used in close proximity to each other.
  - When a class is used as a variable their naming convection should follow the variable naming convection. E.g., std::vector<double> imAnArray;

A.2.2 Documentation

- All technical documentation will be done with Doxygen.
- The header file is the preferred location for class documentation and will contain the tags noted in Appendix C.1.
- Each function will have documentation that contains, input, output, returns variables
- Each variable must be commented, preferably inlined.
- Algorithms and schemes will have a citation to its origin. This could either be a journal paper or some internal document that includes its derivation.
- Keywords in comments should be used as needed, e.g.: TODO, XXX, and FIXME. Where XXX and FIXME implies that there is a bug to be fixed while todo implies that a feature should be added but there are no bugs present.

A.2.3 Classes

- Will have one and only one associated source file and header file.
- Variables should be protected and setters and getters are used to set and get their values. In the case when direct access to a variable is needed for speed a friend class can be used to provide direct access to it. This should still be done via a function call, e.g. getPointerCellVolume().

A.2.4 Generic File Stuff

- Less than 80 characters per line
- Each source file should only contain one class.
- Spaces are always used, tabs are forbidden

A.3 Code Compliance

A.3.1 Memory Management

- Arrays that will be used as part of the "core" routine should be allocated during the init routine and not deleted until the end of the simulation. This is to avoid the creation and destruction of large amount of memory as the simulation progresses.
- Smaller arrays, designed to store a few cells (7 cells) worth of data, may be created on the stack at the start of the core and deleted at the end. The amount should be kept small to avoid stack smashes.

A.3.2 Variable/Function encapsulation

- Class members are private with public getter/setter functions
- Variables will be declared in the scope that they are used.
- When possible variables should be initialized when they are declared.
- Do not import an entire namespace, e.g.

```
       python
BAD: from numpy import *
BETTER: from numpy import sqrt
BEST: import numpy as np
         c++
BAD: using namespace std
BETTER: using namesapce std::cout
BEST: std::cout << ‘‘Sample’’;
```

A.3.3 External Libraries

- TURF "core functionality" may not rely upon external libraries. Only exception is MPI and gtest or small libraries that allow for their source code to be embedded in another source code. An example is the base64 encoder/decoder which consists of one header and source code file and the author allowed redistribution.
- When external libraries are used TURF must still be compilable without them. This is accomplished by using #DEFINES. Example is the exodus mesh importer that uses the netcdf libraries.
- Header files associated with external libraries also need to be surrounded by #ifdef.
- Libraries that require copy left copyright statement cannot be used. For example, the GNU license has a copy left copyright statement that requires programs that link to their libraries to all use the GNU license. The GNU license provides an exception to the copy left statement for system libraries such as libc.
- The preferred license for external libraries is the BSD and others like it.

A.3.4 Language standards

- The use of c99 standards is required for TURF versions up to 2017a. Starting with TURF-2017b c11 is the new standard.
- The use of the auto variable from c11 is not recommended and should be limited to cases of incompatibly between Mac and Linux or for very long iterator declarations.

- Compiler specific features may not be used and proper compiler warnings should be used to ensure compliance with c99/c11 standards. This exception to this is CUDA specific directives and #ifdef must be used to partition the code, e.g.

```
#ifdef USE_NVCC
  some_cuda_code
#endif
```

- The target compilers for TURF-2017a are CUDA nvcc 5.5 and gnu g++ 4.7. Starting with TURF-2017b the required CUDA compiler is nvcc 8.0 and gnu g++ 4.9. Newer versions of nvcc or gnu may be used but features introduced in newer versions may not be used. Other compilers such as Intel and clang are not directly supported by AFRL/RQRS, any discovered incompatibilities should be submitted to AFRL/RQRS as bug fixes or fixed by the user and submitted to AFRL/RQRS as a patch.

A.3.5 Errors and Warnings

As mentioned in the previous section, sufficient warnings should be used to alert the programmer when compiler specific function/capabilities are used. A list of suggested compiler flags are:

	compiler	flags
gnu	required 2017a	-std=c++98 -Wpedantic
	required 2017b	-std=c++11 -Wpedantic
	suggested	-Wall -Wextra -Wshadow -Weffc++
intel	required 2017a	-std=c99 (does this work with c++?)
	required 2017b	-std=c++11
	suggested	-Wall -w2 (or -w3)
CUDA	required	-std

Table A.1: A list of compiler flags required and suggested compiler flags. (Work in progress)

The CUDA compilers only supports the c++11 standard check; to check for code compliance it is recommended to compile with the gnu or other compiler that does. The compiler options listed under suggested tend to lead to better, more consistent code and should be used when compiling. That being said these warnings will highlight formatting errors that don't affect the quality of the compiled code and can be safely ignored.

A.4 Software Testing

This section details how the operators in TURF should be tested. There are two main types of tests, unit tests and configuration tests. These tests are run daily to ensure that updates to the code doesn't change already accepted solutions. To automate this process our group uses Jenkins to pull changes, build a new version of TURF, run the test suite and report any failed tests. This last step is crucial to ensure that errors are found early and reduce the time required to fix new bugs. Although these tests can be used to validate the various operators that is not their primary purpose; instead the main purpose is to alert the developers when TURF produces results that differ from previous versions.

Unit tests are used to ensure that a single operator or routine is working correctly. To do this a test is generated using the gtest framework in which the inputs to an operator or simulated and selected such that the routine produces a known output. For example when testing a root finding algorithm an equation with known roots could be used. Typically these unit tests are included in the source code and are only compiled into executable if certain compiler flags are set. A unit test template is included at the end of the example file TempalteOp.cu in Section C.2.

```
TEST(MSPDistCombineOp_test_case, two_distributions)
{
// Create a dummy world and domain below it
       World* theWorld = new World();
       LogicalDomain* theDom = new LogicalDomain();
       theWorld->addMember(theDom); // Some of constructors require the parent of theDom

// Particle distribution
       MSPDist* srcdist = new MSPDist();
       MSPDist* dstdist = new MSPDist();
       long MaxNparts = 100;
       int MaxNspecies = 1;
       int Nparts1 = 10; int Nparts2 = 20;
       srcdist->init(theWorld->Materials, MaxNparts, MaxNspecies);
       dstdist->init(theWorld->Materials, MaxNparts, MaxNspecies);
       srcdist->setName("source");
       dstdist->setName("destination");
       srcdist->Nparts = Nparts1;
       dstdist->Nparts = Nparts2;
       for (int id=0; id<dstdist->Nparts; id++){
              (*dstdist->cellID)(id) = id; // 0, 1, 2, 3, ... , 19
       }
       for (int is=0; is<srcdist->Nparts; is++){
              (*srcdist->cellID)(is) = is + Nparts2; // 20, 21, 22, 23, ... , 29
       }

// Init MSPDistCombineOp
       MSPDistCombineOp* combine = new MSPDistCombineOp();
       combine->init( srcdist, dstdist );
       combine->apply();

// Test
       EXPECT_EQ( dstdist->Nparts, Nparts1+Nparts2 );
       for (int id=0; id<dstdist->Nparts; id++){
              EXPECT_EQ( (*dstdist->cellID)(id), id );// 0, 1, 2, 3, ... , 29
       }
}
```

A configuration test combines multiple operators together and are tested together. These tests are designed to ensure that various operators work together correctly and can also be used to test operators that are not easily tested through a unit test. To create one of these tests a full TURF simulation is setup and run and the results are saved to a file. These results are then compared against a known solution. A known simulation could be an analytical solution, a standard test case (e.g. Blastwave, Shu-Osher,...), experimental results, or a simulation previously generated by TURF. An analytical solution is the preferable known solution but is not always possible. The final method, comparison with previous TURF simulations, isn't used to validate the code but to alert the developers when the solution changes.

APPENDIX B

TUTORIAL STYLE GUIDE

Samuel J. Araki

Contents

B.1 Directory

When adding a new tutorial example in `tutorial-TURF`, first make a folder representing the physics problem. Within the directory, there should be two folders with documentation and TURF input files, and they should be named as `Doc_*` and `type_of_solver`.

Examples:
`Heatbath_MS` – `Doc_Heatbath` and `Particle`
`CollisionlessShock` – `Doc_Vlasov` and `Vlasov`

B.2 Title Page

Title and authors should be added with the following command.

- `\title [Short Title]{Full Title}`
- `\author [F. Author]{First_Author, Organization1}`
- `\author [S. Author]{Second_Author, Organization2}`
- `\author [T. Author and F. Author]{Third_Author and Fourth_Author, Organization3}`

The title and authors given in the square and curly brackets are displayed in the header and title page, respectively. The title should be always start with "TURF TUTORIAL", followed by problem name, number, and solver/description.

B.3 General Structure

1. Introduction
2. Description of the problem explaining `world.list`
3. Operations used in `operations.list`. This section should be divided by stages or the operations covered in the tutorial.
4. Results if any
5. Conclusions if any

The list of the operations should be included in the section describing `world.list`.

B.4 Main

B.4.1 Text

- Spell out the entire word, e.g. Helium instead of He. or one-dimension rather than 1D.

B.4.2 Labels

Include labels for all sections, subsection, and tables:

- Section: \label{sec:some_unique_keyword}
- Subsection: \label{subsec:some_unique_keyword}
- Equations: \label{eqn:some_unique_keyword}
- Figure: \label{fig:some_unique_keyword}
- Table: \label{tab:some_unique_keyword}
- Lists: \label{lst:some_unique_keyword}

Lists are a bit different in that you can also provide a list for each item in a list. Also if you do not reference the list elsewhere in your document, don't worry about coming up with a label for it. When referencing them, use \cref command to ensure the format is consistent across the tutorials.

B.4.3 Figures

- Make sure the texts are readable.
- Use descriptive captions.
- Always put figures top or bottom of a page just as in standard journals.

APPENDIX C

CODE TEMPLATES

C.1 Header File

```
/**
 * \file TemplateOp.h
 *
 * \class TemplateOp
 * \ingroup
 * \date Apr 11, 2017
 * \author Your Name
 * \brief
 * \details
 * \note
 *
 * \copyright
 * Produced at the Air Force Research Laboratory, AFRL/RQR
 * All rights reserved.
 * \n\n
 * DISTRIBUTION F. Not STINFO Approved.
 * Further dissemination only as directed by AFRL/RQRS,
 * 5 Pollux Drive, Edwards AFB, CA 93524-7048 or higher DoD authority;
 * premature dissemination, July 2013.
 * \n\n
 * THIS CODE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL ERC INC., THE AIR FORCE RESEARCH
 * LABORATORY, OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
 * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS CODE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

```
#ifndef TEMPLATEOP_H_
#define TEMPLATEOP_H_

#include "Operator.h"
#include "LogicalWorld.h"
#include "LogicalDomain.h"

class TemplateOp: public GSOBJECT_DERIVED(Operator )
public:
// Default constructor
    TemplateOp(){
        world=NULL; dom=NULL; mesh=NULL;
        gbx=Box(make_int3(0,0,0),make_int3(0,0,0));
    };

// Main functions
    virtual void init(map<string,GSObject*>* initMap, Domain* thisDomain, int this_stage);
    virtual void apply();
    HDV virtual ~TemplateOp(){};

// Utility functions (NOT FULLY IMPLEMENTED YET)
    HDV void reconnect(){}
    virtual TemplateOp* cloneType(){return new TemplateOp(*this);}
    HDV virtual size_t my_size(){return 0;}

// Unique integers that correspond to core numbers
    enum{ // TASK_DEFAULT = 0
        TASK_ONE = 1,
        TASK_TWO = 2
    };

// Cores
    HDV inline void core(int dim, int4 idx, float time=0.0, float dt=0.0){ //TASK_DEFAULT
        int pi=idx.x;
        // Calculation on each particle or cell of unstructured mesh
    }
    HDV inline void core1(int dim, int4 idx, float time=0.0, float dt=0.0){ //TASK_ONE
        int i=idx.x;
        int j=idx.y;
        int k=idx.z;
        // Calculation on each cell of 3D structured mesh
    }
    HDV inline void core2(int dim, int4 idx, float time=0.0, float dt=0.0){ //TASK_TWO
        int i=idx.x;
        int j=idx.y;
        int k=idx.z;
        // Calculation on each cell of 3D structured mesh
    }

private:
    World* world; ///< \brief pointer to world object
```

```
    Domain* dom; ///< \brief pointer to local domain
    gSMesh* mesh; ///< \brief Cartesian volume mesh

    Box gbx; ///< \brief Extent that includes both the interior and ghost regions
};
#endif /* TEMPLATEOP_H_ */
```

C.2 Source Code

```
/*
* \file TemplateOp.cu
*
* \class TemplateOp
* \ingroup
* \date Apr 11, 2017
* \author Your Name
*
* \copyright
* Produced at the Air Force Research Laboratory, AFRL/RQRS
* All rights reserved.
* \n\n
* DISTRIBUTION F. Not STINFO Approved.
* Further dissemination only as directed by AFRL/RQRS,
* 5 Pollux Drive, Edwards AFB, CA 93524-7048 or higher DoD authority;
* premature dissemination, July 2013.
* \n\n
* THIS CODE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL ERC INC., THE AIR FORCE RESEARCH
* LABORATORY, OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
* TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS CODE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#include "TemplateOp.h"

void TemplateOp::init(map<string,GSObject*>* initMap, Domain* thisDomain, int this_stage){
// Inputs
    READ_PARAMETER_DEFAULT(NAME,string,inname,"TemplateOp","Default Operation Object Name")
    READ_PARAMETER_DEFAULT(USE_GPU,string,inUseGPU,"FALSE","Use GPU/CUDA for Computation [FALSE]")
    READ_PARAMETER_DEFAULT(VERBOSE,string,inverb,"FALSE","Print Info [FALSE]")
    READ_PARAMETER_DEFAULT(HELP,string,doHelp,"FALSE","Display Help Text [FALSE]")
// End read of parameters

// Set flags and display help if true
    verbose = false;
    if(Jstring(inverb->get()).toUpperCase().contains("T")) verbose = true;
    if(Jstring(doHelp->get()).toUpperCase().contains("T")) this->DisplayHelp();
    if(Jstring(inUseGPU->get()).toUpperCase().contains("T")) setUseGPU(true);
    else setUseGPU(false);

// Return if the domain does not exist or not active
    if(thisDomain==NULL || !thisDomain->isActive()){
        setName(inname->get());
```

```
        return;
    }

// Save the world and domain pointers to class for future reference
    world = thisDomain->getParent()->thisAs(World::WORLD_TYPE);
    dom = thisDomain->thisAs(dom);

// Get SMesh object
    mesh = (gSMesh*) thisDomain->thisAs(LogicalDomain::LOGICAL_DOMAIN_TYPE)->getMesh();
    gbx = thisDomain->thisAs(LogicalDomain::LOGICAL_DOMAIN_TYPE)->getGBx();

// Link the object to the one in the domain
    addLink((GSObject**)&mesh,(GSObject*)mesh);

// Activate if apply() is to be performed. Then, set name in order to access
// this operation later if necessary.
    active = true;
    setName(inname->get());
}

void TemplateOp::apply(){
    ApplyCore3D<>(this,&gbx,TASK_ONE,getWorldTime(),getWorldDt());
    ApplyCore3D<>(this,&gbx,TASK_TWO,getWorldTime(),getWorldDt());
}

//-----------------------------------------------------------------------------------------
/*********************************************
 * GTEST
 *********************************************/
/*
#include "gtest.h"
TEST(TemplateOp_test_case, test_name)
{
// Unit test
}
*/
```

References

[1] J. Fife, M. Gibbons, D. VanGilder, and D. Kirtley, "The development of a flexible, usable plasma interaction modeling system," in *38th AIAA Joint Propulsion Conference*, vol. AIAA 2002-4267, 2002.

[2] J. M. Fife, "Hybrid-PIC modeling and electrostatic probe survey of hall thrusters," PhD Dissertation, Massachusetts Institute of Technology, 1998.

[3] L. Brieda, R. Kafafy, J. Pierru, and J. Wang, "Development of the DRACO code for modeling electric propulsion plume interactions," in *40th AIAA Joint Propulsion Conference*, vol. AIAA 2004-3633, 2004.

[4] M. G. Kapper, "Development of advanced numerical algorithms for modeling complex flows," Master's thesis, San Jose State University, 2005.

[5] ——, "A high-order transport scheme for collisional radiative and nonequilibrium plasma," PhD Dissertation, The Ohio State University, 2009.

[6] H. Le, "Development of a chemically reacting flow solver on the graphic processing unit," Master's thesis, San Jose State University, 2011.

[7] L. Cole, "Combustion and magnetohydrodynamic processes in advanced pulse detonation rocket engines," PhD Dissertation, University of California, Los Angeles, 2012.

[8] H. Le, "Hydrodynamic models for multicomponent plasmas with collisional-radiative kinetics," PhD Dissertation, University of California, Los Angeles, 2011.

[9] R. Martin and J. Koo, *Thermophysics universal research framework - infrastructure release*, Version 1.0, Air Force Research Laboratory (AFRL/RQRS), 2015.

[10] B. Whitlock, *Getting data into visit*, Version 2.0.0, LLNL-SM-446033, Lawrence Livermore National Laboratory, 2010, ch. 5. Instrumenting a simulation code.

[11] M. Bettencourt and A. Greenwood, "Performance improvements for efficient electromagnetic particle-in-cell computation on 1000s of CPUs," *IEEE Transactions on Antennas and Propagation*, vol. 56, no. 8, pp. 2178–2186, Aug. 2008.

[12] H. W. Liepmann and A Roshko, *Elements of Gasdynamics*. Mineola, New York: Dover Publications, 1957.

[13] J. D. J. Anderson, *Modern Compressible Flow with Historical Perspective*, 3rd. New York, New York: McGraw-Hill, 2003.

[14] G. A. Bird, *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*. Oxford: Clarendon Press, 1994.

[15] R. J. Taylor, D. R. Baker, and H. Ikezi, "Observation of collisionless electrostatic shocks," *Physics Review Letters*, vol. 24, no. 5, pp. 206–208, Feb. 1970.

[16] H. Ikezi, R. J. Taylor, and D. R. Baker, "Formation and interaction of ion-acoustic solitons," *Physics Review Letters*, vol. 25, no. 1, pp. 11–14, Jul. 1970.

[17] C. Cheng and G. Knorr, "The integration of the vlasov equation in configuration space," *Journal of Computational Physics*, vol. 22, no. 3, pp. 330–351, Nov. 1976, ISSN: 0021-9991. DOI: 10.1016/0021-9991(76)90053-X. [Online]. Available: http://www.sciencedirect.com/science/article/pii/002199917690053X (visited on 01/17/2013).

[18] J.-M. Qiu and A. Christlieb, "A conservative high order semi-lagrangian WENO method for the vlasov equation," *Journal of Computational Physics*, vol. 229, no. 4, pp. 1130–1149, Feb. 2010, ISSN: 0021-9991. DOI: 10.1016/j.jcp.2009.10.016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999109005610 (visited on 01/17/2013).

[19] U. Ayachit, *The ParaView guide*, version Version 5.0, Avalable at www.paraview.org, Kitware Inc., 2015, 233 pp.

[20] *VisIt user's manual*, version Version 1.5, UCRL-SM-220449. Avalable at https://visit.llnl.gov, Lawrence Livermore National Laboratory, 2005, 356 pp.

[21] T Blacker, S. J. Owen, M. L. Staten, *et al.*, *Cubit: Geometry and mesh generation toolkit. 15.2 user documentation.* SAND2016-1649, Sandia National Laboratory, 2016, 976 pp.

[22] Y. Yamamura and N. I. N. Matsunami, "Theoretical studies on an empirical formula for sputtering yield at normal incidence," *Radiation Effects*, vol. 71, 1983.

[23] N. Matsunami, Y. Yamamura, Y Itikawa, N Itoh, Y Kazumata, S Miyagawa, K Morita, R Shimizu, and H Tawara, "Energy dependence of the ion-induced sputtering yields of monatomic solids," *Nuclear Data Tables*, vol. 31, pp. 1–80, 1984.

[24] Y Yamamura and H Tawara, "Energy dependence of the ion-induced sputtering yields of monatomic solids at normal incidence," *Nuclear Data Tables*, vol. 62, pp. 149–253, 1996.

[25] "Computational modeling of a hall thruster plasma plume in a vacuum tank," Master's thesis, Massachusetts Institute of Technology, 2002.

[26] K Kannenberg, V Khayms, B Emgushov, L Werthman, and J Pollard, "Validation of hall thruster plume sputter model," in *37th Joint Propulsion Conference*, AIAA 2001-3986, Salt Lake City, Utah, 2001.

[27] J.-F. Roussel, J Bernard, and Y Garnier, "Numerical simulation of induced environment, sputtering, and contamination of satellite due to electric propulsion," in *Proceedings Second European Spacecraft Propulsion Conference*, Aug. 1997, pp. 517–522.

[28] Y Garnier, V Viel, J.-F. Roussel, and J Bernard, "Low-energy xenon ion sputtering of ceramics investigated for stationary plasma thrusters," *Journal of Vacuum Science and Technology A*, vol. 17, no. 6, pp. 3246–3254, Nov. 1990.

[29] E. J. Pencil, T Randolph, and D Manzella, "End-of-life stationary plasma thruster far-field plume characterization," in *32nd Joint Propulsion Conference*, AIAA 1996-2709, Lake Buena Vista, Florida, 1996.

[30] Z. L. Zhang and L Zhang, "Anisotropic angular distributions of sputtered atoms," *Radiation Effects and Defects in Solids*, vol. 159, pp. 301–307, 2004.

[31] M. K. Scharfe, J. W. Koo, and G. Azarnia, "DSMC implementation of experimentally-based Xe^+ + Xe differential cross sections for electric propulsion modeling," in *AIP Conference Proceedings*, vol. 1333, AIP, 2011, pp. 1085–1090.

[32] J Greenwood, "The correct and incorrect generation of a cosine distribution of scattered particles for monte-carlo modelling of vacuum systems," *Vacuum*, vol. 67, pp. 217–222, 2002.